Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs

Sasho Nedelkoski*, Jasmin Bogatinovski*, Alexander Acker*, Jorge Cardoso†, Odej Kao*
*Distributed and Operating Systems, TU Berlin, Berlin, Germany
{nedelkoski, jasmin.bogatinovski, alexander.acker, odej.kao}@tu-berlin.de
†Huawei Munich Research Center, Huawei Technologies, Munich, Germany
jorge.cardoso@huawei.com

***Abstract*—The detection of anomalies is an essential data mining task for achieving security and reliability in computer systems. Logs are a common and major data source for anomaly detection methods in almost every computer system. Recent studies have focused predominantly on one-class deep learning methods on manually specified log representations. The main limitation is that these models are not able to learn log representations describing the semantic differences between normal and anomaly logs, leading to a poor generalization on unseen logs. We propose Logsy, a classification-based method to learn log representations that allow to distinguish between normal system log data and anomaly samples from auxiliary log datasets, easily accessible via the internet. The idea behind such an approach to anomaly detection is that the auxiliary dataset is sufficiently informative to enhance the representation of the normal data, yet diverse to regularize against overfitting and improve generalization. We perform several experiments on publicly available datasets to evaluate the performance and properties, where we show improvement of 0.25 in F1 compared to previous methods.**

***Index Terms*—anomaly detection, log data, transformers, systems reliability**

I. Introduction

Anomaly detection [1] is a data mining task of finding observations in a corpus of data that differ from the expected behaviour. Anomalies in large systems such as cloud and high-performance computing (HPC) platforms can impact critical applications and a large number of users [2]. Therefore, a timely and accurate detection is necessary for achieving reliability, stable operation, and mitigation of losses in a complex computer system.

Logs are an important data source for anomaly detection in computer systems [3]–[5]. They are free-form text that represent interactions between data, files, services, or applications, and are typically utilized by developers, and data-driven methods to understand system behaviours and to detect, localize, and resolve problems that may arise.

A common approach for data-driven log anomaly detection is one-class classification [6]. Thereby, the objective is to learn a model that describes the normal system behaviour, usually assuming that most of the unlabeled training data is non-anomalous. The detection is realized by labelling samples that lie outside of the learned decision boundary as anomalies. The massive log data volumes in large systems have renewed the interest in the development of one-class deep learning methods to extract general patterns from non-anomalous samples. Previous studies have been focused mostly on the application of long short-term memory (LSTM)-based models [4], [5], [7]. They leverage log parsing [8], [9] on the normal log messages and transform them into log templates, which are then utilized to train sequence learning models.

However, the learning of the sequence of templates still have limitations in terms of generalization for previously unseen log messages. The methods tend to produce false predictions mostly owing to the imperfect log vector representations. For example, learning sequence of indices [4] fails to correctly classify a newly appearing log messages. When pre-trained word vectors are utilized [5], [7], the domain where the word vectors are pre-trained (e.g., Wikipedia) has essential differences from the language used in computer system development.

Often, the assumption for the normal data in anomaly detection methods is that it should be compact [10]. This means the normal log messages should have vector representations with close distances between each other, e.g. concentrated within a tight sphere, and the anomalies should be spread far from the distribution of normal samples. We propose a new anomaly detection method that directly addresses the challenge of obtaining representative and compact numerical log embeddings. We train a neural network to learn log vector representations in a manner to separate the normal log data from the system of interest and log messages from auxiliary log datasets of other systems. The concept of such a classification approach to anomaly detection is that the auxiliary dataset helps to learn a better representation of the normal data while regularizing against overfitting. This ultimately leads to a better generalization on unseen logs. For example, for a target system logs of interest T where anomaly detection needs to be performed, as auxiliary data could be employed one or more datasets from an open-source log repository (e.g., [11]). As a neural network architecture, we adopt the Transformer encoder with multi-head self-attention mechanism [12], which learns context information from the log message in the form of log vector representations (embeddings). In addition, we propose a new hyperspherical learning objective enforces the model to learn compact log vector representations of the normal log messages. This enforces for the normal samples to have concentrated (compact) vector representations around the

2374-8486/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDM50108.2020.00148

centre of a hypersphere. It enables better separation between the normal and anomaly data, where a distance from the centre of such a sphere is used to represent an anomaly score. Small distances correspond to normal samples, while large distances correspond to anomalies.

The contributions of this study can be summarized in the following points.

1) A new classification-based method for log anomaly detection utilizing self-attention and auxiliary easy-accessible data to improve log vector representation.
2) Modified objective function using hyperspherical decision boundary, which enables compact data representations and distance-based anomaly score.
3) The proposed approach is evaluated against three real anomaly detection datasets from HPC systems, Blue Gene/L, Thunderbird, and Spirit. The method significantly improves the evaluation scores compared to those in the previous studies.

II. Related Work

A significant amount of research and development of methods for log anomaly detection has been published in both industry and academia [4], [5], [7], [8], [13], [14].Supervised methods were applied in the past to address the log anomaly detection problem. For example, [13] applied a support vector machine (SVM) to detect failures, where labels for both normal and anomalous samples are assumed to be available. For an overview of supervised approaches to log anomaly detection we refer to Brier et al. [15]. However, obtaining system-specific labelled samples is costly and often practically infeasible.

Several unsupervised learning methods have been proposed as well. Xu et al. [14] proposed using the Principal Component Analysis (PCA) method, where they assume that there are different sessions in a log file that can be easily identified by a session-id attached to each log entry. The publicly available implementation allows for the term frequency-inverse document frequency (TF-IDF) representation of the log messages, utilized in our experiments as a baseline. Lou et al. [16] proposed Invariant Mining (IM) to mine the linear relationships among log events from log event count vectors.

The wide adoption of deep learning methods resulted in various new solutions for log-based anomaly detection. Zhang et al. [17] used LSTM to predict the anomaly of log sequence based on log keys. Similar to that, DeepLog [4] also use LSTM to forecast the next log event and then compare it with the current ground truth to detect anomalies. Vinayakumar et al. [18] trained a stacked-LSTM to model the operation log samples of normal and anomalous events. However, the input to the unsupervised methods is a one-hot vector of logs representing the indices of the log templates. Therefore, it cannot cope with newly appearing log events.

Some studies have leveraged NLP techniques to analyze log data based on the idea that log is a natural language sequence. Zhang et al. [17] proposed to use the LSTM model and TF-IDF weight to predict the anomalous log messages. Similarly, LogRobust [5] and LogAnomaly [7] incorporate pre-trained word vectors for learning of a sequence of logs where they train an attention-based Bi-LSTM model.

Different from all the above methods, we add domain bias on the anomalous distribution to improve detection [19]. We provide such bias by employing easily accessible log datasets as an auxiliary data source.

III. Towards Classification-Based Log Anomaly Detection

Anomaly detection can be viewed as density level set estimation [20]. Steinwart et al. [19] state that this can be interpreted as binary classification between the normal and the anomalous distribution and point out that the bias on the anomalous distribution is essential for improved detection. Meaning that if we provide some information to the model of how anomalous data looks like, it will boost its performance. For instance, we may interpret the class assumption that semi-supervised anomaly detection methods require on the anomalies, as such prior knowledge [10]. Moreover, specific types of data can have inherent properties that allow us to make more informed prior assumptions such as the word representations in texts [21]. In our proposed work the assumption is that each word meaning depends on its context. We assume that drawing samples from some auxiliary easy-access corpus of log data can be much more informative for an added description of normal and anomalies compared to sampling noise, or no auxiliary data.

A. Preliminaries

We define a log as a sequence of temporally ordered unstructured text messages $L = (x_i : i = 1, 2, ...)$, where each message x_i is generated by a logging instruction (e.g. printf(), log.info()) within the software source code, and i is its positional index within the sequence. The log messages consist of a constant and an optional varying part, respectively referred to as log template and variables.

The smallest inseparable singleton object within a log message is a token. Each log message consists of a finite sequence of tokens, $\mathbf{r_i} = (w_j : w_j \in \mathbb{V}, j = 1, 2, ..., |r_i|)$, where $\mathbb{V}$ is a set (vocabulary) of all tokens, j is the positional index of a token within the log message x_i, and $|r_i|$ is the total number of tokens in x_i. For different x_i, $|\mathbf{r_i}|$ can vary. Depending on the concrete tokenization method, w_j can be a word, word piece, or character. Therefore, tokenization is defined as a transformation function $\mathcal{T} : x \rightarrow \mathbf{r}$.

With respect to our proposed method, the notions of context and numerical vector representation (embedding vector) are additionally introduced. Given a token w_j, its context is defined by a preceding and subsequent sequence of tokens, i.e. a tuple of sequences: $C(w_j) = ((w_1, w_2, ..., w_{j-1}), (w_{j+1}, w_{j+2}, ..., w_{|\mathbf{r_i}|}))$, where $0 \leq j \leq |\mathbf{r_i}|$. An embedding vector is a d-dimensional real valued vector representation $\mathbf{s} \in \mathbb{R}^d$ of either a token or a log message.

B. Problem Definition

Let $\mathcal{D} = \{(\mathbf{S_1}, y_1), \ldots, (\mathbf{S_n}, y_n)\}$ be a set of training logs from the system of interest where $\mathbf{S_i} \in \mathbb{R}^{d \times |r_i|}$ is a log message. Its tokens are represented in $d-dimensional$ space, i.e. the log message is represented by $d \times |r|$ matrix, where $|r|$ is number of tokens and $y_i = 0; 1 < i \leq n$, assuming that the data in the system of interest is mostly composed of normal samples. Let $\mathcal{A} = \{(\mathbf{S_{n+1}}, y_{n+1}), \ldots, (\mathbf{S_{n+m}}, y_{n+m})\}$, where m is the size of the auxiliary data and $y_i = 1; n < i \leq n+m$. Let $\phi(\mathbf{S_i}, y_i, \theta) : \mathbb{R}^d \rightarrow \mathbb{R}^{d \times |r_i|}$ be a function represented by a neural network, which maps the input log message token embeddings to vector representations $\mathbf{z} \in \mathbb{R}^p$, and $l : \mathbb{R}^p \rightarrow [0, a], a \in \mathbb{R}$ be a function, which maps the output to an anomaly score. The task is to learn the parameters θ from the training data, and then for each incoming instance in the prediction phase $\mathcal{D}_t = \{(\mathbf{S_1^t}), (\mathbf{S_2^t}), \ldots\}$, t indicates test sample, predict whether it is an anomaly or normal based on the anomaly scores obtained by $l(\phi(\mathbf{S_i}, y_i, \theta))$.

IV. Self-Attentive Anomaly Detection with Classification-based Objective

We refer to the data from the system of interest as target dataset, i.e. the system where we want to detect anomalies. Important to note is that we are not using any anomaly data from the target system for learning purposes in our experiments. The term auxiliary data refers to other non-related systems, which serve only for training the model. All the results during test time are performed on a test set extracted from the target dataset.

A. Logsy

The method is composed of two main parts, the tokenization of the log messages and the neural network model. In the following section, we discuss the inner workings of the proposed method, which is depicted in Fig. 1.

Tokenization. Tokenization transforms the raw log messages into a sequence of tokens, as shown in Fig. 1. For this purpose, we utilize the standard text preprocessing library NLTK [22]. The message is first filtered for HTTP and system path endpoints (e.g., /p/gb2/stella/RAPTOR/). Lower case transformation of all characters is applied, and all of the ASCII special characters are removed. The log message is split into word tokens. We remove every token that contains numerical characters, as they often represent variables in the log message. Additionally, we remove the most commonly used English words that are in the stop words dictionary of NLTK (e.g., *the* and *is*). To the front of the tokenized log message, a special '[EMBEDDING]' token is added. In the model, the '[EMBEDDING]' token attends over all original tokens from the sample, which enables the model to summarize the context of the log message in the vector representation. All tokens from every log message form vocabulary $\mathbb{V}$ of size $|\mathbb{V}|$, where each token is represented with integer label $i \in 0, 1, \ldots, |\mathbb{V}| - 1$. An advantage of Logsy compared to previous approaches is that it does not depend on log parsers as a pre-processing step. We consider the tokenized log message as direct input to the model. There is no loss of information from the log message, due to the imperfections that exist in the log parsing methods.

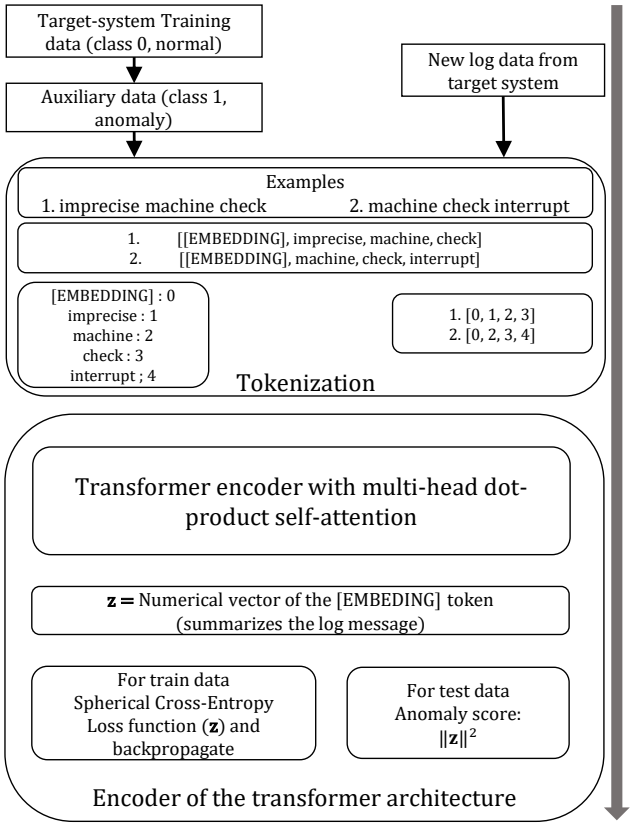

Fig. 1. Overview of the architecture and component details of Logsy.

Model. Logsy has two operation modes – offline and online. During the offline phase, log messages are used to tune all model parameters via backpropagation and optimal hyper-parameters are selected. During the online phase, every log message is passed forward through the saved model. This generates the respective log vector representation $\mathbf{z}$ and an anomaly score for each message.

The model applies two operations on the input tokens: token vectorization (token embeddings) and positional encoding. The subsequent structure is the encoder of the Transformer [12] module with multi-head self-attention, which takes the result of these operations as input. At the output of the encoder, there are $|r_i|$ transformed vector representation from the initial tokens. Each vector has the dimensionality d. This also represents the size of all the layers of the model and the word embeddings. The representation obtained from the 'EMBEDDING' token is utilized in the proposed objective function. In other words, we train the model such that the log representations from the normal data are compact with close distances, while the anomalies are distant. The last two parts are the objective (loss) function during training and the computation of the anomaly score for test samples. Based on

the loss, gradients are back-propagated to tune the parameters of the model, while based on the anomaly score we decide if the sample is anomalous or normal. In following we describe the objective function.

B. Objective function

To ensure learning of the intrinsic differences of normal and anomaly log samples, we propose a spherical loss function. It is designed to integrate the previously mentioned assumption that normal data is often concentrated having close distances between the normal samples, while also learning properties to distinct from anomalous samples. This is done by employing a radial classification loss which enforces a compact hyper-spherical decision region for the normal samples.

To derive the loss, we start with the standard binary cross entropy. Let $\mathcal{D} \cup \mathcal{A} = \{(\mathbf{S_1}, y_1), \ldots, (\mathbf{S_{n+m}}, y_{n+m})\}$ be the concatenation of the training logs from the system of interest and the auxiliary data with $\mathbf{S_i} \in \mathbb{R}^{d \times |r_i|}$. $y_i \in \{0, 1\}$, and $y_i = 0$ denotes normal samples (target system), while $y_i = 1$ denotes an anomaly (auxiliary data). Let $\phi(\mathbf{S_i}, \theta) : \mathbb{R}^d \to \mathbb{R}^p$ be our encoder architecture that maps the token embeddings form the log message to a $p - dimensional$ vector. Let $l : \mathbb{R}^p \to [0, 1]$ be a function which maps the output to an anomaly score. Using $\phi(\mathbf{S_i}, \theta)$ and $l(\cdot)$, the standard binary cross-entropy loss can be written as:

$$-\frac{1}{n}\sum_{i=1}^{n}(1-y_i)\log l(\phi(\mathbf{S_i};\theta)) + y_i \log(1 - l(\phi(\mathbf{S_i};\theta))) \quad (1)$$

In the standard binary classifier with sigmoid function, the decision boundary is half-space. The representation of the log messages is not guaranteed to be compact in this case. It could be very possible that the normal samples are scattered through the space with varying, potentially very large distances between them. To enforce compactness of the representations of the log messages we utilize the Gaussian radial basis function as $l(\cdot)$:

$$l(\mathbf{z}) = \exp(-\|\mathbf{z}\|^2) \quad (2)$$

Replacing the function into the loss function we get the hyper-spherical classifier:

$$\begin{aligned} &\frac{1}{n}\sum_{i=1}^{n}(1-y_i)\|\phi(\mathbf{S_i};\theta)\|^2 \\ &-y_i \log(1 - \exp(-\|\phi(\mathbf{S_i};\theta)\|^2)) \end{aligned} \quad (3)$$

This ensures compactness of the normal samples, which is enforced to be around the center of a sphere $\mathbf{c} = \mathbf{0}$. For normal samples, i.e., $y_i = 0$, the loss function will minimize the distance to $\mathbf{c}$. This results in low values for the left term in Equation 3. In contrast, the right term of the loss function favors large distances for the anomalous samples. The center of a sphere c could be any constant value, which is not relevant during the optimization.

C. Anomaly score and detecting anomalies

Considering that the assumption of the objective function enforces compactness of representations, i.e. samples are close to the center of the sphere $\mathbf{c} = \mathbf{0}$, we define our anomaly score as the distance of the log vectors (obtained from the 'EMBEDDING' token) to the center $\mathbf{c}$ of the hypersphere.

$$A(\mathbf{x_i}) = \|\phi(\mathbf{x_i};\theta)\|^2 \quad (4)$$

We define low anomaly scores $A(\mathbf{x_i})$ to be normal log messages, while large scores are anomalies. To decide if the sample is anomalous or normal, we use a threshold $\mathcal{E}$. If the anomaly scores $A(\mathbf{x_i}) > \mathcal{E}$, then the sample is an anomaly, otherwise, we consider it as normal.

V. Evaluation

To quantify the performance of Logsy, we perform a variety of experiments. We compare the method against two publicly available baselines DeepLog and PCA on three real-world HPC log datasets. We describe the main properties of the datasets, discuss the experimental setup, and present the results. Logsy is evaluated against unsupervised approaches, as from the perspective of using labels of the target system, it is an unsupervised approach.

A. Experimental setup

We select three open real-world datasets from HPC systems for evaluation as target systems, namely Blue Gene/L, Spirit, and Thunderbird [11]. They share an important characteristic associated with the appearance of many new log messages in the timeline of the data, i.e. the systems change over time. Furthermore, as an additional dataset for enriching the auxiliary data in all experiments we use the HPC RAS log dataset [23]. Due to the absence of labels this dataset cannot be used for evaluation purposes–can not be a target dataset.

For each target dataset we use logs from the remaining datasets as auxiliary data to represent the anomaly class. It is important to note that the target vs auxiliary splits, ensure that there is no leak of information from the target system into the auxiliary data. Meaning, there are no labeled samples from the target system into the auxiliary data. These logs originate from other IT systems. The non-anomalous samples from the target system are the target dataset. For example, when Blue Gene/L is our target system proportion of the negative samples of Thunderbird, Spirit, and RAS are used as an auxiliary dataset to represent the anomaly class. These auxiliary samples could be also error messages obtained from online code repositories (e.g., GitHub). We perform anomaly detection on the test samples from the target dataset for determining the scores.

Several experiments are conducted with different train-test splits on varying target datasets. To ensure that the test data contains new log messages previously unseen in the training we always split the data when sorted by the timestamp of the log messages. As described in Table I, we perform 5 different data splits to cover as many possible scenarios.

TABLE I
DATASET DETAILS.

System	#Messages	#Anomalies	#Anomalies5m	#Unique Log messages in test and not in train for every split					total unique messages
				10%	20%	40%	60%	80%	
Blue Gene/L	4747963	348460	348460	2679	2621	2256	2231	465	4486
Thunderbird	211212192	3248239	226287	334	127	71	27	12	3279
Spirit	272298969	172816564	764890	1091	1028	297	129	73	3441

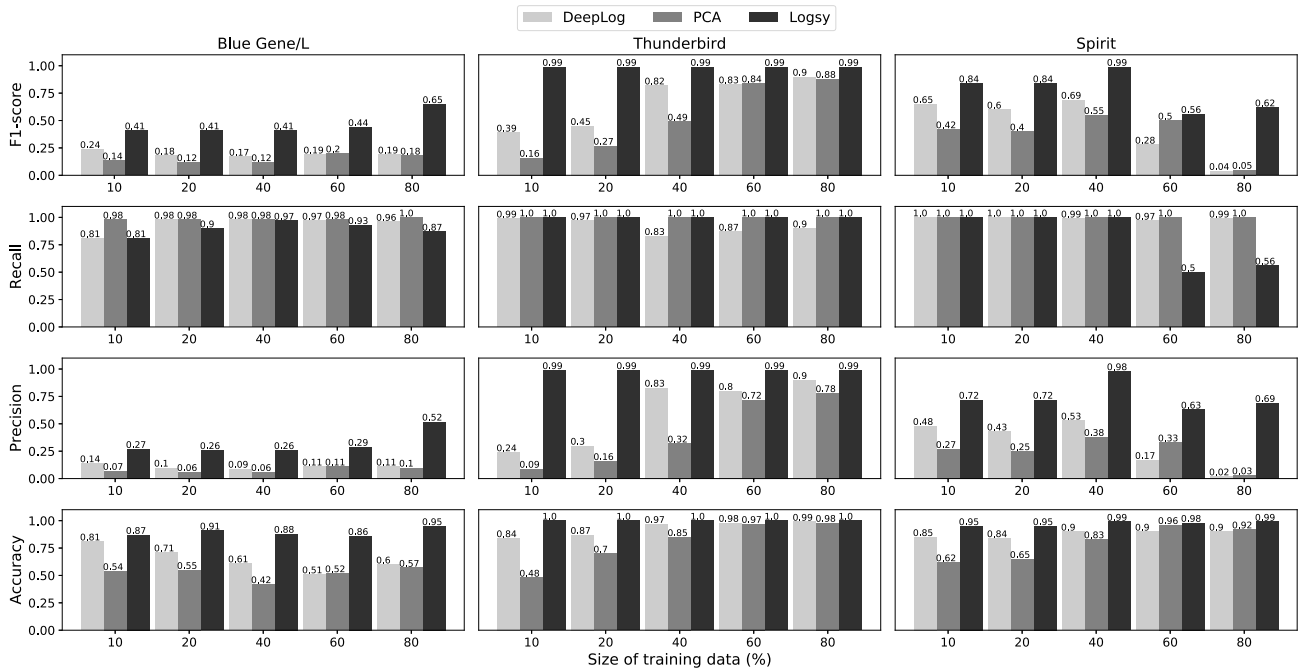

Fig. 2. Comparison of the evaluation scores against the two baselines DeepLog and PCA on three different datasets.

1) Baselines: We compare Logsy against two publicly available baseline methods, i.e. PCA [14] and Deeplog [4]. The current claimed state-of-the-art method LogAnomaly [7] to best of our knowledge has no publicly available implementation, as it is industry-related research. Moreover, LogAnomaly reports only marginal improvement over DeepLog of 0.03 F1 score, and thus both approaches are relatively comparable. The parameters of these methods are all tuned to produce their best F1 score.

2) Logsy: Implementation details: Every log message during tokenization is truncated to a maximum of $\max(|r_i|) = 50$ tokens. Logsy has two layers of the transformer encoder, i.e. $N = 2$ in Fig. 1. The tokens are embedded with 16 neural units, and the higher level vector representations obtained with the transformer encoding are all of the same sizes. The size of the feed-forward network that takes the output of the multi-head self-attention mechanism is also 16, which makes the '[EMBEDDING]' vector has the same size. For the optimization procedure for every experiment, we use a dropout of 0.05, Adam optimizer with a learning rate of 0.0001, and a weight decay of 0.001. We address the imbalanced number of normal versus anomaly samples with adding weights to the loss function for the two classes, 0.5 for the normal and 1.0 for the anomaly class. The models are trained until convergence and later evaluated on the respective test split.

B. Results and discussion

We show the overall performance of Logsy compared to the baselines in Fig. 2. Generally, Logsy achieves the best scores, having an averaged F1 score in all the splits of 0.448 on the Blue Gene/L dataset, 0.99 on the Thunderbird dataset, and 0.77 on the Spirit data. Both DeepLog and PCA, have lower F1 scores in all experiments performed. It is shown that the baselines have a very high recall, but also low precision. This means they can find the anomalies, however, producing large amounts of false-positive predictions. Logsy, on the other hand, preserves the high recall across the datasets and evaluation scenarios but shows a large improvement in the precision scores. This is due to the correct classification of new unseen log messages and the reduction of the false positive rate. For instance, on the Blue Gene/L dataset, DeepLog and PCA respectively show 2-4 times lower precision compared to Logsy. Overall, Logsy is the most accurate method having an average of 0.9. If a log anomaly detection method generates

too many false alarms, it will add too much overhead to the operators and a large amount of unnecessary work. Therefore, high precision methods are favourable. DeepLog leverage the indexes of log templates, which ignore the meaning of the words in the log messages, to learn the anomalous and normal patterns. However, different templates having different indexes can share common semantic information and both could be normal. Ignoring this information results in the generation of false positives for DeepLog compared to Logsy.

We notice that increasing the training size also increases the F1 score in almost all methods, except for the last two splits in Spirit. These splits are unfortunate as they have a very small number of anomalies. Important to note is that Logsy outperforms the baselines even when only 10% of the data is training data. For example, in Blue Gene/L we have 0.32 F1-score on 10% training data, while the largest F1-score of the baselines is 0.24. In Thunderbird, this difference is even more noticeable, where an F1-score of 0.99 is already achieved in with the first 10%. This shows that even with a small amount of training data from the target system, Logsy extracts the needed information of what causes a log message to be normal or anomaly, and produces accurate predictions even in unseen samples.

VI. Conclusion

In this work a new anomaly detection approach, called Logsy, is presented. It achieves improved generalization abilities on new, unseen log samples, compared to related approaches. The model is based on a self-attention encoder network with a hyperspherical classification objective. We formulated the log anomaly detection problem in a manner to discriminate between normal training data from the system of interest and samples from auxiliary easy-access log datasets from other systems, which represent an abnormality. Experimental evidence was presented showing that our classification-based method performs well for anomaly detection. The results of our method outperformed the baselines by a large margin of 0.25 F1 score.

We believe that future research on log anomaly detection should focus on finding alternative ways to incorporate richer domain bias emphasising the diversity of normal and anomaly data.

References

[1] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 1–21, 1969.

[2] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, and Z. Zang, "Rapid and robust impact assessment of software changes in large internet-based services," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–13.

[3] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[4] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.

[5] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[6] M. M. Moya, M. W. Koch, and L. D. Hostetler, "One-class classifier networks for target recognition applications," *NASA STI/Recon Technical Report N*, vol. 93, 1993.

[7] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs."

[8] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," 2020.

[9] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[10] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," *arXiv preprint arXiv:1906.02694*, 2019.

[11] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[13] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 2007, pp. 583–588.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[15] J. Breier and J. Branišová, "Anomaly detection from log files using data mining techniques," in *Information Science and Applications*. Springer, 2015, pp. 449–457.

[16] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection."

[17] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, "Automated it system failure prediction: A deep learning approach," *2016 IEEE International Conference on Big Data (Big Data)*, pp. 1291–1300, 2016.

[18] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 236–242, 2017.

[19] I. Steinwart, D. Hush, and C. Scovel, "A classification framework for anomaly detection," *Journal of Machine Learning Research*, vol. 6, no. Feb, pp. 211–232, 2005.

[20] A. B. Tsybakov *et al.*, "On nonparametric estimation of density level sets," *The Annals of Statistics*, vol. 25, no. 3, pp. 948–969, 1997.

[21] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[22] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[23] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "Co-analysis of ras log and job log on blue gene/p," in *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 840–851.