Machine Learning based Anomaly Detection of Log Files using Ensemble Learning and Self-Attention

Markus Fält*†, Stefan Forsström*, Tingting Zhang*

*Mid Sweden University,
Department of Information Systems and Technology
Sundsvall Sweden, Holmgatan 10, 852 30
{markus.falt, stefan.forsstrom, tingting.zhang}@miun.se

†Försäkringkassan,
ITPBM Systems Management
Sundsvall Sweden, Södra Järnvägsgatan 41, 852 37
markus.falt@forsakringskassan.se

***Abstract*—Modern enterprise IT systems generate large amounts of log data to record system state, potential errors, and performance metrics. Manual analysis of log data is becoming more difficult as these systems become more complex. Therefore, machine learning based anomaly detection of system logs is a vital component for the future of system management. Existing log anomaly detection models commonly rely on learning the general normal behavior of the target systems to accurately detect anomalies. They are however limited by the often sparse existing system knowledge. Therefore, this paper proposes a general anomaly detection method which requires little or no knowledge of the target system. This is done by assuming there are semantic similarities in different systems' log data. Labeled log data from other systems can then be used for training the anomaly detection model. The model uses self-attention transformers and ensemble learning techniques to learn the semantic representation of normal and abnormal log messages. The proposed method achieves a performance comparable to other log anomaly detection methods while requiring little knowledge of the target system.**

***Index Terms*—log, anomaly detection, AIOps, attention, ensemble learning, machine learning**

I. Introduction

AIOps (Artificial Intelligence Operations) [1], [2] is a new term used in the IT industry for automated system management. AIOps is described by using artificial intelligence to simplify IT operations management and accelerate and automate problem resolution in complex modern IT environments. One critical problem with implementing AIOps solutions is to automatically detect anomalies. Anomaly detection is a well studied area but it is often difficult to create unsupervised methods for this, especially for anomaly detection of system logs.

As computer systems become more complex and interconnected, so does the data generated by these systems. With an increasing amount of data generated by system logs, error reports, performance metrics, etc, there is now a trend in system management of automating system analysis.

Large scale IT systems have many different connected components, often developed by many different teams with different logging/error-handling practices. This can make root-cause analysis and error detection difficult for system administrators, not only because of the wide range of logging data but also because of the amount of data that is logged by the systems. It is rare that system logs only contain errors, often logs generated by a system include both general system behavior, performance metrics, and unexpected anomalous events. In the case of large-scale failures or errors it can be difficult to separate the normal expected behavior from the anomalous unexpected behavior that is represented in the system logs. System experts may be able to define rules for doing this separation, this can be thought of as a manual definition of what represents normal system behavior. Defining such rules requires expert knowledge of the system and is not generalizable to other systems, therefore generalizable solutions based on deep learning or other machine learning methods may be useful.

The main challenge of anomaly detection has to do with the unbalanced nature of the problem. The amount of data that represents the normal or expected behavior is much greater than the amount of data that represents the unexpected or abnormal behavior. This means that when training a machine learning model we cannot use accuracy as a measure of performance. Also a normal loss function may not converge to a useful state very quickly. This problem can be dealt with in various ways, such as: artificially extending the negative class samples or using a modified loss function that gives a higher value to predicting the negative samples correctly.

This paper attempts to examine the effect semantic representation of log statements have on predicting expected normal events or unexpected anomalous events. A log statement will usually contain a semantic section describing the event in human readable text, this is written by a system developer and is often meant to be read by other system developers. In order to find semantic-similarities or -differences between expected and unexpected log events, Nedelkoski, Bogatinovski, Acker, *et al.* [3] introduces a method called Logsy. The Logsy method relies on the fact that errors from auxiliary systems can be used as examples of unexpected behavior. This makes it possible to extend the number of negative samples; however, a similar method should also be able to produce generic log anomaly detection models that do not need information about the target system, and are thus completely unsupervised with regard to the target system.

This paper attempts to take this one step further, for a more generalized system that does not need to rely on data from the target system. Thus, this paper aims to present a log anomaly detection method that is completely trained on auxiliary system data and unsupervised with regard to the target system. All in order to answer the following 3 research questions:

1) How reliable is the presented method at detecting system log anomalies in terms of recall and precision?
2) What affect does ensemble learning solutions have on the performance of the log anomaly detection method?
3) How does the performance and potential scalability issues affect the feasibility of the method when applied to real-world systems?

From these research questions, the primary contribution in this article is to present and evaluate a log anomaly detection method that is able to be trained with zero knowledge about the target system.

The remainder of this article is organized as follows: Section II present more background to the problem and relevant related work. Section III presents our method and an overview of the whole proposed system. Section III-E presents our experimental setup. Section IV presents our results and measurements. Section V presents our discussion and evaluation of the results. Finally, Section VI presents our conclusions and future work from this research.

II. Background

Many log anomaly detection methods try to identify useful templates of log messages, and then try to either find anomalies in the sequences of log message templates or in a combination of the template meaning and the template sequences. Parsing and structuring log data is often a requirement when it comes to preparing log data for automatic handling. This can mean, finding a set structure for log messages, identifying log message templates, and extracting logged parameters and variable values.

Research in log parsing has produced a number of methods [4]–[9] that rely on identifying log message templates. These methods will attempt to identify constant and variable parts of the messages, a pattern matching string based on the identified constant pattern will then be used to correctly identify different types of messages.

A. Related work

Du, Li, Zheng, *et al.* [10] present an unsupervised method DeepLog for training an anomaly detection model, the training method only requires normal data from the target system. The method relies totally on the sequences of log message types, and identifies uncommon sequences by predicting the next log message type in the sequence. If an unexpected message type appears the sequence is labeled as an anomaly. The method is shown to be robust and generalizes well to different log data sets.

Farzad and Gulliver [11] present an anomaly detection method that uses auto-encoder neural networks together with an isolation forest. More specifically two auto-encoders are used, the first one is trained to extract features from the pre-processed log data, the second one is used to determine a threshold value for finding anomalies. This method is shown by the authors to perform very well on the used testing data sets. The method is simple, effective and can be trained by using only normal log data if needed. The authors also examine how using only the isolation forest method would work and conclude that it is not suitable for log data. This is likely due to high dimensionality in the pre-processed log data.

Huang, Liu, Fung, *et al.* [12] present HitAnomaly a supervised anomaly detection method for system log data. HitAnomaly combines the semantic meaning and the sequence of log events, it also considers the sequence of parameter values and applies an attention mechanism both to the parameter values and template sequences to classify the anomalies. This is done by using a hierarchical transformer structure. The method was shown to be robust and outperformed other methods on the tested data.

Nedelkoski, Bogatinovski, Acker, *et al.* [3] present Logsy an anomaly detection method that uses self-attention transformers. Logsy relies on the assumption that anomalous messages from unrelated log sources often are semantically similar to anomalous messages of the target log source. Logsy shows promising results by achieving as good or better performance compared to DeepLog. It is however worth noting that Logsy will not be able to take parameter values into account in the anomaly detection. This means any anomalous logged parameter values will not be considered. This is because Logsy works solely on the semantic representations of the log messages. Logsy can however correctly identify anomalous log messages that have never been seen, DeepLog cannot do this. This is because DeepLog relies on the log template keys extracted during the log parsing process.

Xia, Zhang, and Jiang [13] present a novel approach for log anomaly detection, where they simply use the collective results of other anomaly detection algorithms. This can also be referred to as the mixture of experts approach, where each anomaly detection method's output is put through a gating network and used in combination to form a decision. The results of the two ensemble methods presented by Xia, Zhang, and Jiang show that ensemble methods can provide much better performance than any of the tested anomaly detection methods can on their own.

III. Method

The amount of data generated by IT systems has increased in recent years, this has led to a need for increased automation and smart monitoring of IT systems. Automatic detection of system errors has become a necessity due to the time requirements for manual log analysis.

Nedelkoski, Bogatinovski, Acker, *et al.* [3] presents a method called Logsy, for anomaly detection of log data. Logsy uses additional log data sources to supplement the negative training samples, so that the model will learn the system's normal behavior better. Inspired by Logsy, this article attempts to explore the possibility of creating generalized models for deviation detection of log data. This article tries to see if generalized models trained on only additional log data sources can be successfully applied to unseen log data sources.

A. Data sets

To train and evaluate the performance of the proposed method, three labeled data sets are used: Blue Gene/L, Thunderbird, and Spirit [14]. The collected log data was retrieved from LogHub [15], an open collection of log data sets that can be used in research.

1) Blue Gene/L: The BGL (Blue Gene/L) data set contains log data collected from a supercomputer at Lawrence Livermore National Labs in California.

2) Thunderbird: The Thunderbird data set contains log data collected from a supercomputer at Sandia National Labs in Albuquerque New Mexico.

3) Spirit: The Spirit data set is also from a HPC (High-Performance Computing) cluster similar to BGL and Thunderbird.

B. Preprocessing

To prepare the log data for training and testing, only the semantic part of each log event was extracted. The semantic representation of a log event was produced by removing everything from the log text that is not English words. The first step in doing this is to identify a common structure in each log source.

Example of a BGL log event and its associated structure and extracted "Content":

```
Log Event: - 1117845034
2005.06.03 R20-M0-N7-C:J15-U01
2005-06-03-17.30.35.039771 R20-M0-N7-C:J15-U01
RAS KERNEL INFO total of 14 ddr error(s)
detected and corrected
Log Structure: <Label> <Timestamp> <Date>
<Node> <Time> <NodeRepeat> <Type> <Component>
<Level> <Content>
Content: total of 14 ddr error(s) detected and
corrected
Tokens: total of ddr error s detected and
corrected
```

The "Content" section was prepared by removing all special characters and any tokens that contain numbers. Next each token was mapped to a unique number and all token sequences were padded to a length of 50.

C. Model structure

The overall model structure can be divided into three parts word embedding, attention mechanism, and output.

The purpose of word embedding is to embed a kind of relative semantic meaning with each word. This is usually done by converting words into n-dimensional vectors. The idea is that words with similar meanings are assigned similar vectors, while words that are opposite to each other are given vectors that differ. This type of word embedding allows a model to learn a deeper semantic meaning, instead of just learning common word sequences.

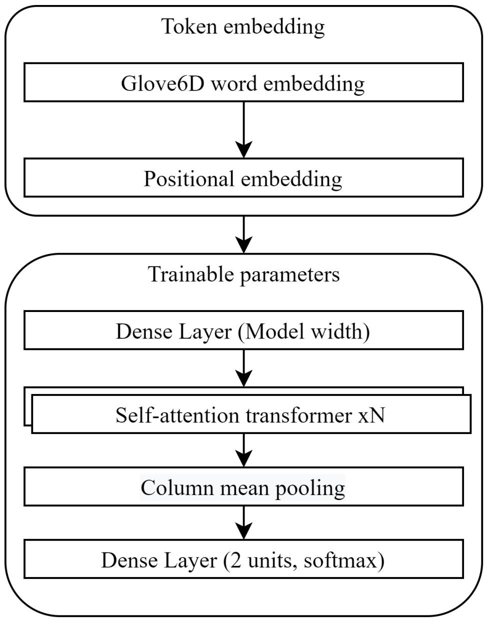

Figure 1. Model structure overview.

The next step in the model is an attention mechanism that learns how different words relate to each other. The method for this is usually called self-attention transformers and was first presented in an article by Vaswani, Shazeer, Parmar, *et al.* [16]. The article describes how this method can be successfully used for language translation through an encoding/decoding structure. The method can also be generalized well in other machine learning areas.

The column mean pooling calculates the mean value for each column in the output from the self attention transformer. This is done both to reduce the number of parameters and to flatten the data before the dense output layer.

Last is the output of the model, which is a 2-class softmax dense layer, the reason for this is so that different weights can be assigned in the loss function depending on the target class.

Figure 1 shows an overview of the model structure and how it can be split into trainable and non-trainable sections. The model width that is referenced in figure 1 is the internal dimensionality of the model while N is the number of self-attention transformers that are stacked in sequence. We can think of N as being the "depth" of the model, both the model width and depth are hyper-parameters that can be tuned.

1) Word-embedding: Word embeddings take words and convert them into n-dimensional vectors, a sequence of words then becomes a sequence of vectors, this can be viewed as a matrix. The number of dimensions that was used for n is 50, the word sequences were also padded to a length of 50. The output of the word embedding is therefore a 50x50 matrix.

The word embedding that is used is a pre-trained word embedding database created using a method called GloVe [17]. In this case, the word embedding database used was trained on wikipedia text with about 6 billion words and with a dimensionality of 50. The reason for using a pre-trained word

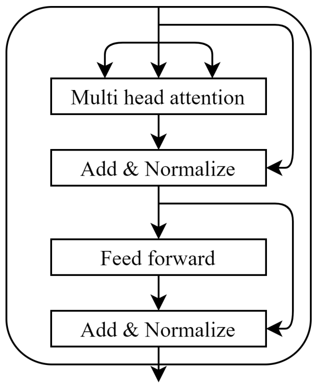

Figure 2. Self-attention transformer.

embedding is that the number of parameters that needs to be optimized in the model decreases. The majority of parameters are often used for word embedding, by removing these we reduce the training time.

As can be seen in figure 1, the word embedding part contains not only the GloVe embedding but also a positional embedding. The positional embedding is required as self-attention transformers can in no way distinguish the order of words. Therefore, we need to encode this information on top of the GloVe embedding. This is done by adding p to each value in the GloVe embedding according to equation 1.

$$\begin{aligned} p_{j,2k} &= sin(\frac{j}{10000^{\frac{2k}{n}}}) \\ p_{j,2k+1} &= cos(\frac{j}{10000^{\frac{2k}{n}}}) \end{aligned} \tag{1}$$

In equation 1, j represents row, $2k$ and $2k+1$ represents even and odd columns, and n represents the dimensionality of the glove embedding.

2) Dimension reduction: The data that is output from the word embedding has a form of 50x50, this requires an internal dimensionality of 50 in the self-attention transformer layer. However, the model uses a linear transformation from 50x50 to 50x8 (where 8 is the model width), the linear transformation is done through a dense layer with linear activation function. The reason for reducing the dimensionality in this way is to avoid over fitting and to reduce the amount of trainable parameters.

3) Attention mechanism: In order for the model to learn the relationships between different word vectors, self-attention transformers are used. Self-attention transformers are specially created to learn the semantic context between all the words in a sequence. Figure 2 shows the structure of a self-attention transformer, as can be seen there are two major components: the multi-head attention block and the feed forward network.

Addition and normalization is done on the output and input of both the multi-head attention block and the feed forward network. The addition and normalization with the input makes it so that the input pattern is not lost by the transformer.

The input data for the multi-head attention block is split into multiple "attention heads", the purpose of having multiple attention heads is to let each attention head have a unique point of view of the input data. Each attention head will capture different semantic factors of the input data.

For each attention head i in the multi-head attention block there are three trainable weight matrices $\mathbf{W}_K^{(i)}$, $\mathbf{W}_Q^{(i)}$, and $\mathbf{W}_V^{(i)}$. The three weight matrices are multiplied by the input matrix $\mathbf{X}$ (see equation 2) that contains the vector representation of each word in a sequence. Since the word vectors are not processed in a sequence but rather all at the same time, it is important that the order of words are encoded into the word vector representations.

As can be seen in equation 2 three new matrices $\mathbf{K}_i$, $\mathbf{Q}_i$, and $\mathbf{V}_i$ are created for each attention head. These matrices are referred to as key, query and value respectively.

$$\begin{aligned} \mathbf{K}_i &= \mathbf{X} \cdot \mathbf{W}_K^{(i)} \\ \mathbf{Q}_i &= \mathbf{X} \cdot \mathbf{W}_Q^{(i)} \\ \mathbf{V}_i &= \mathbf{X} \cdot \mathbf{W}_V^{(i)} \end{aligned} \tag{2}$$

The output $\mathbf{X}_i'$ from one attention head is calculated using equation 3. As can be seen the product of the query and the transposed key matrices are divided by the square root of w (number of dimensions for each attention head), this is to scale down the output of the product before the softmax. The query and key product can be seen as a type of attention score, and by applying the softmax function this score is turned into a probability distribution. By then multiplying by the value we get the attention output for the i:th attention head.

The exact same steps are done in parallel for each attention head the only differences are the initial weight matrices $\mathbf{W}_K^{(i)}$, $\mathbf{W}_Q^{(i)}$, and $\mathbf{W}_V^{(i)}$.

$$\mathbf{X}_i' = softmax(\frac{\mathbf{Q}_i \cdot \mathbf{K}_i^T}{\sqrt{w}}) \cdot \mathbf{V}_i \tag{3}$$

The final multi-head attention output is calculated by concatenating all attention head outputs together, then multiplying by a trainable matrix $\mathbf{W}_C$ to reduce the dimensionality to the same as the input. The multi-head attention output $\mathbf{X}'$ is described by equation 4, where m is the number of attention heads.

$$\mathbf{X}' = concat(\mathbf{X}_1', \mathbf{X}_2', ..., \mathbf{X}_m') \cdot \mathbf{W}_C \tag{4}$$

The feed forward network consists of two linear transformations with a relu activation function in-between. The output of the feed forward network is combined with its input that previously passed through the multi-head attention block. This combined output is then the output of the self-attention transformer.

There are two dropout layers that are not visible in figure 2, they are applied directly to the multi-head attention output and the feed forward output before the addition and normalization.

4) Loss function: The output of the self-attention transformer has the shape 50x8, this needs to be converted to a shape of 1x2 since there are two classes to predict. Each row in the output of the self-attention transformer can be thought of as the context of a single word. So to get the average context for all words we can average the columns of the output, this will give a new shape of the data that is 1x8. This is then put through a dense softmax layer with 2 units.

The loss function that is used is weighted categorical cross-entropy. The reason to weigh the two classes differently is that anomalies in the testing data are uncommon, so we need to value predicting anomalies higher than predicting normal. By biasing the model towards predicting anomalies, then force it to learn to recognize normal events we can achieve a better performance.

Equation 5 shows the loss function used where t_i is the true label, $\phi(x_i;\theta)$ is the prediction generated by the model, and w_i is the weight of t_i.

$$WCE = -\sum_{i}^{2} t_i log(\phi(x_i;\theta))w_i \tag{5}$$

There are two target classes that the model should learn, normal and anomaly. The weight for normal w_1 is 0.1 and the weight for anomaly w_2 is 0.9. This distribution reflects the approximate inverse distribution of how many normal examples exist for each anomaly example.

D. Ensemble-learning

An ensemble learning technique is used to make the method more robust. While training the individual transformer models it was noticed that the there was a large variance in model performance. Because of this a bagging ensemble learning method was used, where each base model would vote on a prediction. The prediction votes were then counted and multiplied by a weight value. Finally, the prediction with the highest prediction score is chosen.

E. Experimental Setup

Table I
HYPER-PARAMETER LIST

Hyper-parameter	Value
Number of models	100
Number of epochs	15
Batch size	256
Word sequence length	50
Model width	8
Model depth	1
Drop out rate	0.05
Learning rate	0.0001
Loss weight normal	0.1
Loss weight abnormal	0.9

Testing the performance of the model for the three data sets was done by training the model using data from two of the data sets and evaluating the model on the remaining data set. For each data set one hundred self-attention transformer models were trained. Each of the self-attention transformer models were trained on 100,000 randomly selected samples from the two training data sets. The sampling was random for each trained model, because of this the training data for each model was slightly different. The goal is to create many weak log anomaly detectors and through ensemble learning create a model that is greater than the sum of its parts. Table I shows the parameters that were used in training the base models.

To test the ensemble learning model each self-attention transformer model will vote on a prediction for each sample in the target data set. This will produce two lists of votes for the entire target data set, one for the normal votes and one for the anomalous votes. The votes have to be weighed differently because of the bias in the models. To find a "best" weight a linear search is done to find the weight with the highest f1-score. This gives us a possible best case performance but we can also try to approximate the vote weights to find a weight that can give a good f1-score without having to rely on being able to see the test labels.

IV. RESULTS

To judge the quality of the proposed model three metrics were used: precision (PPV), recall (TPR), and f1-score.

$$PPV = \frac{TP}{TP + FP} \tag{6}$$

$$TPR = \frac{TP}{TP + FN} \tag{7}$$

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} \tag{8}$$

Table II
BGL RESULTS

	Normal	Anomaly
Precision	0.97	0.46
Recall	0.95	0.54
F1	0.96	0.50

The BGL results that can be seen in table II, show that the model was able to correctly predict slightly more than half of the anomalies. We can also see that slightly more than half of the predicted anomalies was in reality normal log messages. The BGL data set contained approximately 7% anomalies.

The Tbird results that can be seen in table III show that the model was able to perform nearly perfect. It should be noted that the values in all result tables have been rounded to two

Table III
TBIRD RESULTS

	Normal	Anomaly
Precision	1.0	1.0
Recall	1.0	1.0
F1	1.0	1.0

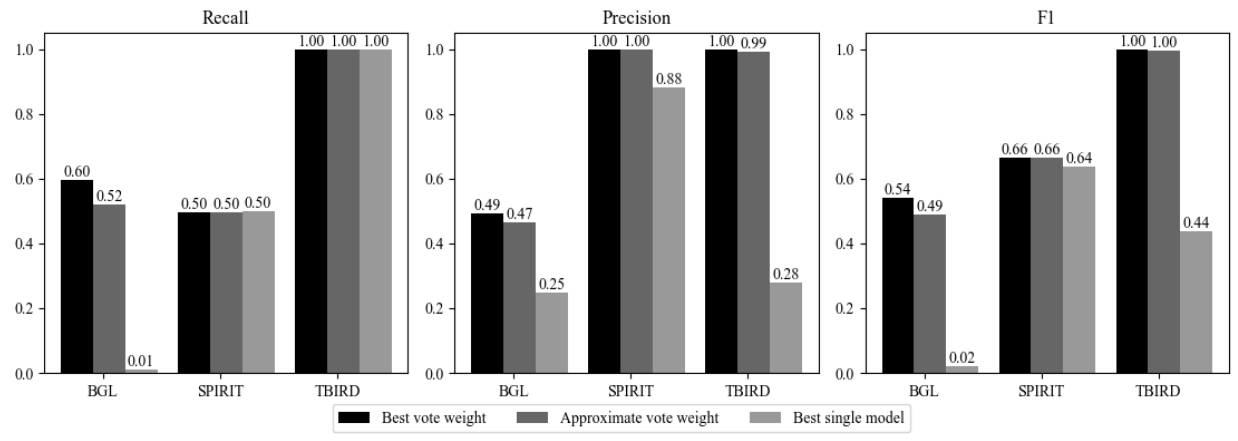

Figure 3. Comparison of model performance.

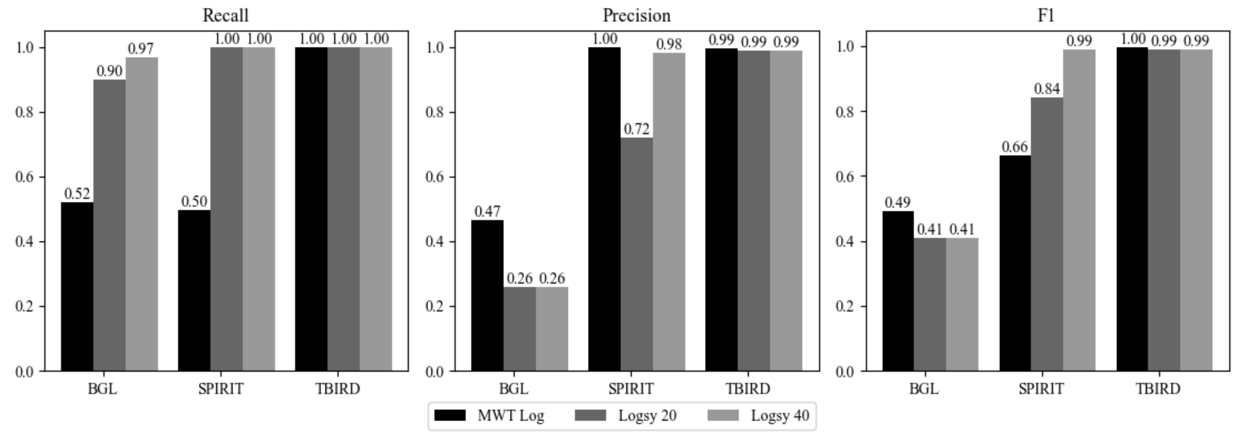

Figure 4. Compare with Logsy.

Table IV
SPIRIT RESULTS

	Normal	Anomaly
Precision	0.91	1.0
Recall	1.0	0.49
F1	0.96	0.66

decimal points, this means that all values in table III were above 0.995.

The Spirit results shown in table IV show that the model is able to accurately predict anomalies as the precision is very high. However only about half of the anomalies are correctly classified.

Figure 3 shows a comparison of the performance of the ensemble learning model and the best single model. The ensemble learning model was tested by evaluating the vote weights in two ways. First was to find the optimal weights by searching for the weight distribution with the highest f1-score. Second was to estimate the proportion of anomalous messages in the data set and choose the weight distribution that is closest to predicting a similar amount of anomalies, this can be seen as the “approximate vote weight” in figure 3. All of the tested data sets contain approximately 7% anomalies so 7% was used to get the approximate vote weight result.

Figure 4 shows how the proposed model compares to a similar model Logsy [3]. The figure shows the performance of Logsy trained both with 20% and 40% of the target data. Many weak transformers (MWT) Log refers to the method proposed by this article and in figure 4 MWT Log uses the same “approximate vote weight” method as mentioned for figure 3.

V. DISCUSSION

The proposed method was able to perform well on the tested data sets compared to similar log anomaly detection methods. However, the anomalies that this method can detect are limited, only anomalies that are visible by the semantics of the log messages can be detected. This means that anomalies in the sequence of events cannot be detected as well as anomalies in parameter values.

The work was compared to Logsy [3] because it uses a similar method and was the inspiration for this work. When training Logsy, data from the target system as well as data from auxiliary systems are used. This makes it possible for Logsy to learn the behavior of the target system well. MWT Log does not use data from the target system for training, and instead relies completely on labeled log data from auxiliary systems. In figure 4 it can be seen that MWT Log achieves a slightly higher f1-score for both the BGL and TBIRD data sets compared to Logsy. Considering no data from the target data sets were used in training for MWT Log and that it was able to in some cases perform better than Logsy shows that the method could generalize well. MWT Log may be more generalize because of this, however MWT Log is also likely to be much slower with both training and predicting. The reason MWT Log is much slower is because of the ensemble learning technique that is used. In a real real world system caching may mitigate the slow prediction process somewhat by reducing the number of predictions.

The reason for applying an ensemble learning technique was because of the uncertainty of the trained model. When relying on only auxiliary data for training it was difficult to predict how well the trained model would perform on the target test data. However, by training multiple models and allowing each to contribute to a decision the accuracy and precision of the predictions became better.

VI. Conclusions

This work presents a log anomaly detection method that uses multiple weak transformers to learn the general semantic representation of an anomaly.

Our first investigated research question was about the reliability in terms of recall and precision of the presented system. For which a method was presented and tested, the tested performance of the method was comparable to other similar log anomaly detection methods. The second research question was about the effect ensemble learning would have on the performance. For this the ensemble learning method was compared one of the best single models that was trained. It showed that for two data sets ensemble learning contributed significantly to the performance over a single model. Finally, in regards to both of these research questions we were able to show that the proposed method performs well. Considering no data from the target data set was used in the training process the method was still able to achieve comparable performance to other similar state-of-the-art solutions. With regards to research question 3 the scalability of the system might be a limiting factor for its usage. This is because of the ensemble technique used which requires many models for performing a single prediction. Depending on if the model is deployed such that multiple single models can perform predictions concurrently will affect the speed of predictions. To help mitigate these issues it may be possible to cache predictions in order to minimize the number of predictions.

The work shows that there are some semantic similarities between the tested data sets that can be helpful for detecting anomalies. The work attempts to show that semantic information from different systems can be used to train general anomaly detectors. The performance of the method was only tested on three labeled data sets, more testing is however required to better judge the method's performance.

Only anomalies that are shown in the semantics of the log-messages can be caught. The context of a log message and its logged parameters contain useful information about the state of a system.

The focus when continuing this work will be to investigate how the context and logged parameters can be used to help improve the performance of log-anomaly detectors.

References

[1] *IBM AIOps*, https://www.ibm.com/cloud/learn/aiops.

[2] Y. Dang, Q. Lin, and P. Huang, "Aiops: Real-world challenges and research innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, 2019, pp. 4–5.

[3] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *2020 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2020, pp. 1196–1201.

[4] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, IEEE, 2016, pp. 859–864.

[5] 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.

[6] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, IEEE, 2018, pp. 167–16 710.

[7] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*, IEEE, 2003, pp. 119–126.

[8] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 1255–1264.

[9] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *2015 11th International conference on network and service management (CNSM)*, IEEE, 2015, pp. 1–7.

[10] 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*, 2017, pp. 1285–1298.

[11] A. Farzad and T. A. Gulliver, "Unsupervised log message anomaly detection," *ICT Express*, vol. 6, no. 3, pp. 229–237, 2020.

[12] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[13] X. Xia, W. Zhang, and J. Jiang, "Ensemble methods for anomaly detection based on system log," in *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, IEEE, 2019, pp. 93–931.

[14] 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.

[15] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[17] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162.