Leveraging Log Instructions in Log-based Anomaly Detection

Jasmin Bogatinovski*, Gjorgji Madjarov‡, Sasho Nedelkoski*, Jorge Cardoso† and Odej Kao*
* Technical University Berlin, Berlin, Germany, Email: jasmin.bogatinovski@tu-berlin.de
† Huawei Munich Research, Munich, Germany
‡ University Ss Cyril and Methodius, Skopje, North Macedonia

2022 IEEE International Conference on Services Computing (SCC) | 978-1-6654-8146-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/SCC55611.2022.00053

***Abstract*—Artificial Intelligence for IT Operations (AIOps) describes the process of maintaining and operating large IT systems using diverse AI-enabled methods and tools for, e.g., anomaly detection and root cause analysis, to support the remediation, optimization, and automatic initiation of self-stabilizing IT activities. The core step of any AIOps workflow is anomaly detection, typically performed on high-volume heterogeneous data such as log messages (logs), metrics (e.g., CPU utilization), and distributed traces. In this paper, we propose a method for reliable and practical anomaly detection from system logs. It overcomes the common disadvantage of related works, i.e., the need for a large amount of manually labeled training data, by building an anomaly detection model with log instructions from the source code of 1000+ GitHub projects. The instructions from diverse systems contain rich and heterogenous information about many different normal and abnormal IT events and serve as a foundation for anomaly detection. The proposed method, named ADLILog, combines the log instructions and the data from the system of interest (target system) to learn a deep neural network model through a two-phase learning procedure. The experimental results show that ADLILog outperforms the related approaches by up to 60% on the F_1 score while satisfying core non-functional requirements for industrial deployments such as unsupervised design, efficient model updates, and small model sizes.**

***Index Terms*—anomaly detection, log data, system dependability, AIOps, deep learning**

I. Introduction

IT infrastructures commonly consist of thousands of networked software (microservices) and hardware (e.g., IoT, Edge) components. The uninterrupted and correct interaction is crucial to overall system functionality. However, this IT complexity combined with the required QoS guarantees (e.g. maximal latency) increasingly overwhelms the IT operators in charge. The current trends of agile software development with hundreds of updates and daily deployments further exacerbate the operational challenges. The holistic overview, operation, and maintenance of the IT infrastructure grow even more challenging when additionally it is affected by unforeseen factors such as failures, security breaches, or external environmental events. Companies react to these threats by employing additional site reliability engineers (SREs) as well as by deploying AI-enabled methods for IT operations (AIOps) [1].

The AIOps methods collect and analyse plenty of IT system information – metric data, logs, and traces to detect anomalies, locate their root causes and remediate them. The diverse AIOps techniques enable fast, efficient and effective prevention of upcoming failures, aiming to minimize their hazardous effects during the daily operational activities [1].

In this paper, we focus on anomaly detection in the context of AIOps, as a core step towards enhancing fault tolerance: the earlier an anomaly is detected, the more time is available to prevent the failure and mitigate the impact on the QoS. We focus on system log messages (logs) as semantically rich data written by humans for humans. Logs are generated from log instructions (e.g., log.info(”VM took %f seconds to spawn.”, createSeconds)). They visualise important system events and give hints for the operators that run the system as a black-box [2]. The log instructions are commonly composed of static text (log template), parameters of the event (e.g., createSeconds), and log level giving information about the severity level of the event (e.g., ”info”, ”fatal”, ”error”). The log levels come at different granularities. The lower log levels such as ”info” are commonly used when describing *normal* state or state transitions, e.g., ”Successful connection.”. In contrast, higher levels such as ”error”, ”critical”, or ”fatal” commonly accompany events that describe *abnormal* states or state transitions, e.g., ”Machine failure”. Therefore, the log levels encode rich expert information for detecting anomalies, and they are frequently used in today’s operational practices [2]. For example, to diagnose an anomaly, operators search for logs with higher levels such as ”critical”, or ”fatal” [3].

Owning to the complexities of the IT systems, logs are constantly generated in large volumes (e.g., up to several TB per day [4]). The emergence of complexity makes the manual log-based anomaly detection time-consuming [2], prompting the need for automation [5], [6]. Thereby, automatic methods for log-based anomaly detection are increasingly researched and adopted [7]–[10]. Current methods are commonly grouped into two families, i.e., supervised and unsupervised [5]. Existing supervised methods depend on manually labeled training data. Due to the constant evolution of the software systems [9], the supervised methods require a repetitive, **time-expensive labeling process**, which is oftentimes practically challenging and infeasible [4]. The unsupervised methods mitigate the labeling problem by modeling with logs from normal system states and detecting any significant deviations from the modeled normality state as anomalies. However, the lack of explicit information about anomalous logs during modeling leads to **limited input representation**, reducing the detection performance, and questioning their practical usability [5].

To address the two challenges, we propose ADLILog. The central idea of the method is to use data from public code projects (e.g., GitHub) alongside the data from the system of interest (*target system*) when learning the anomaly detection model. Since the public code projects have many log instructions for diverse events, we assume that they may encode rich anomaly-related information. Following the usage of the log levels for log anomaly detection, we considered grouping the instructions based on their levels to extract anomaly-related information. Specifically, we created two severity level groups from the instructions based on their levels – ”normal” (composed of ”info”) and ”abnormal” (”error”, ”fatal”, and ”critical”). To verify our assumption, we conducted a study to examine the anomaly-related language properties between the two groups (i.e., diversity in the vocabulary and the sentiment of the words). The study results show that the two groups extract anomaly-related information that can be used as a basis for anomaly detection. Based on this observation, we introduce ADLILog, which uses the anomaly-related information alongside the target system data to learn a deep learning anomaly detection model through a two-phase learning procedure. We extensively evaluate ADLILog against three related methods on two widely used benchmark datasets and demonstrate that our method outperforms the supervised methods by 5-24%, and the unsupervised by 40-63% on the F_1 score.

The remaining of the paper is structured as follows. Section II presents our study that examines the potential of the log instructions to aid anomaly detection. Section III introduces ADLILog. Section IV gives the experimental results. Section V discusses the related work. Section VI concludes the paper and gives directions for future work.

II. Examining the potential of log instructions for log-based anomaly detection

In this section, we examine the potential of the log instructions to aid anomaly detection. We start with our observation that there exist two log instructions severity groups, based on their log levels, i.e., ”normal” (”info”) and ”abnormal” (”fatal”, ”critical”, and ”error”). We assume that the static texts of the instructions have complementary properties preserving anomaly-related information. To study the validity of the assumption, we analyze two language properties of the n-grams from the static texts concerning the two groups.

A. Log Instruction Collection and Processing

For the starting point of the analysis, we created a dataset of log instructions from the source code of more than 1000 public code projects from GitHub. We included a wide spectrum of domains and programming languages (Python, Java, C++), covering different log instruction types. The heterogeneity enables us to examine the vocabulary diversity and the semantic properties of diverse normal and abnormal events across systems. To account for the reliability of the log level assignment, we selected projects with more than a 100-stars and at least 20 contributors.

We process the log instructions by extracting the log levels and the static texts. The diverse programming languages use different names for the log levels. As a first step, we unify all the log levels. We preprocess the static texts by applying several operations, including lower-case word transformation, splitting the static texts on whitespace, removing placeholders, and removing ASCII special characters and stopwords from the Spacy English dictionary [11]. We refer to this data as **Severity Level (SL)** data. It is a set of tuples from two elements – (1) the static text of log instruction and (2) the severity group (e.g., (”machine error”, ”abnormal”)). We used this data for the log instruction examination study. Similar to related studies [12], we extracted the n-grams from the static text by varying *n* in the range $n = \{3, 4, 5\}$.

B. Log Instructions Static Texts Uniqueness Analysis

Intuitively, when describing abnormal events, the static text typically contains n-grams like ”failure” or ”error connection”, as opposed to normal events, where n-grams like ”successful” and ”accepted” are more likely to appear. Therefore, we assume that the log instructions static texts of the two severity level groups share partially overlapping vocabularies. To verify this, we considered an approach from information theory that defines the amount of information uncertainty in a message [13]. In our case, we analyze the relation of the n-grams with the two severity groups. At first, given an n-gram (e.g., ”machine failure”), there is high uncertainty for the assigned severity group. As we receive more information about the n-gram (e.g., new logs with the n-gram ”machine failure”), its uncertainty concerning the associated severity group is reduced. To measure the uncertainty, we used Normalized Shanon’s entropy [13]. We calculated the entropy for each n-gram and reported the key statistics of the n-grams distribution.

TABLE I
Log Instructions Static Texts Uniqueness Analysis Results

	Min	1st Qu.	Median	3rd Qu.	Max
Average Entropy	0.00	0.00	0.00	0.27	0.51

TABLE I summarizes the results. It is seen that the median of the distribution is 0. This means that the majority of the n-grams are associated with only one of the two severity groups. Thereby, *the two severity groups are characterized with a rather unique vocabulary.* While this analysis gives information about the uniqueness of the vocabularies, it does not account for the type of expressed intent. Thereby, we made an n-gram sentiment analysis (where the sentiment is used to quantify the intent type, i.e., positive or negative).

C. Log Instructions Static Texts Sentiment Analysis

To evaluate the n-gram sentiment concerning the two severity groups, we considered a pretrained sentiment analysis model from Spacy [11]. We run the n-grams through the model to obtain the sentiment score. We used the sentiment score to categorize the n-grams into three categories, i.e., positive, negative and neutral. We relate the events from the

TABLE II
LOG INSTRUCTIONS STATIC TEXTS SENTIMENT ANALYSIS RESULTS

Sentiment	Positive			Negative			Neutral		
Severity Group	Normal	Abnormal	Shared	Normal	Abnormal	Shared	Normal	Abnormal	Shared
N-gram Coverage [%]	66.94%	28.13%	4.93%	23.13%	69.75%	7.12%	46.98%	43.43%	9.59%

”normal” severity group with positive intent because they describe a successful state or state transition. Similarly, we relate the ”abnormal” group with negative intent. The third category contains n-grams with neutral intent, i.e., events without strongly expressed intent.

TABLE II summarize the results. For each of the three sentiment categories, we show the percentages of the n-grams concerning the two severity groups. In the positive intent category 66.94% of the n-grams are associated with the normal, and 28.13% are related to the abnormal severity group. In contrast, from the n-grams associated with negative intent, 69.75% are associated with the abnormal group, 23.13% are associated with the normal severity group. Therefore, *there exists a relationship between the normal group and positive intent, and the abnormal group and the negative intent*. The proposed severity log level grouping aligns with human intuition when expressing positive and negative sentiments. By combining the two observations, we conclude that the SL data has rich anomaly-related properties.

III. ADLILOG: LOG-BASED ANOMALY DETECTION BY LOG INSTRUCTIONS

Following our observations from the examination study, in this section, we introduce ADLILog as an unsupervised log-based anomaly detection method. Fig. 1 illustrates the overview of the approach. Logically, it is composed of (1) *log preprocessing*, (2) *deep learning framework* and (3) *anomaly detector*. The role of the *log preprocessing* is to process the raw logs by carefully selecting preprocessing transformations that expose rich information for the deep learning framework. The *deep learning framework*'s goal is to learn and output useful log representations for the target-system logs. It does so by training a deep neural network model with a sequential two-phase learning process (pretraining and finetuning), during which data from the target-system logs and the SL data are used. The *anomaly detector* detects if the input target-system logs are normal or anomalous. In the following, we describe the three components of ADLILog.

A. Log Preprocessing

The raw target-system logs are characterized by high noise which affects the anomaly detection performance [5]. To that end, we preprocess the logs by removing all path endpoints (e.g., /bin/) and split the static text using whitespaces into singletons we call tokens. The tokens with numeric values often denote parameters. We consider them as noise and remove them. Similar to the preprocessing for the SL data, we apply the same set of preprocessing operations. In addition, each log is prepended with a dedicated Log Message Embedding ([LME]) token. The [LME] token is an important design detail. We use it to extract a numerical representation of the log from the neural network, further given as input to the detector. Finally, we introduce a hyperparameter max_len to unify the lengths, as the network requires fixed-size input.

B. Deep Learning Framework

The deep learning framework consists of three components: 1) embedding layer, 2) encoder network from Transformer architecture [14] and 3) classification layers. Given the tokenized logs at the input, the embedding layer transforms the input tokens into numerical vector – embeddings. We then use the encoder network to learn relationships between the vector embeddings and the appropriate target. The output from the encoder layer is the vector embedding of the input log/(static text), i.e., the [LME] vector. Depending on the training phase (pretraining or finetuning), the [LME] vector proceeds towards one of the two classification layers. The output from the classification layers is used as input in the appropriate loss function. After finetuning, the output from the second set of classification layers is the final vector embedding of the input log, which proceeds towards the anomaly detector.

1) Embedding Layer: The *embedding layer* receives the preprocessed logs as input. It serves as an interface between the textual and numerical token representation format. Specifically, each token is assigned a single index corresponding to a token embedding vector. The embeddings are learned during pretraining. The embeddings are learned jointly with the parameters of the neural network. Notably, the embedding layer is updated just during the pretraining phase.

2) Log Message Encoder: As a suitable architecture for the log message encoder, we identified the encoder of the Transformer [14]. This architecture provides state-of-the-art results in many NLP tasks (e.g., sentiment analysis) [15]. By pointing to the similarities between the log static texts and natural language [12], we justify our design of choice. The encoder implements a multi-head self-attention mechanism that exploits the relations between tokens within the static texts. At the output of the encoder, we provide the vector embedding of the [LME] token. Due to the architectural design, the vector of the [LME] token attends over all the other token vectors, allowing efficient feature extraction.

3) Classification Layers: The *classification layers* as input receive the [LME] token from the encoder. It is composed of two sets of linear neural layers. As depicted in Fig. 1, the first layer set (*Set 1*) has two linear layers, with parameters θ'. It is trained jointly with the encoder during the pretraining procedure. The output of the first set of classification layers is given towards the binary cross-entropy as a loss function

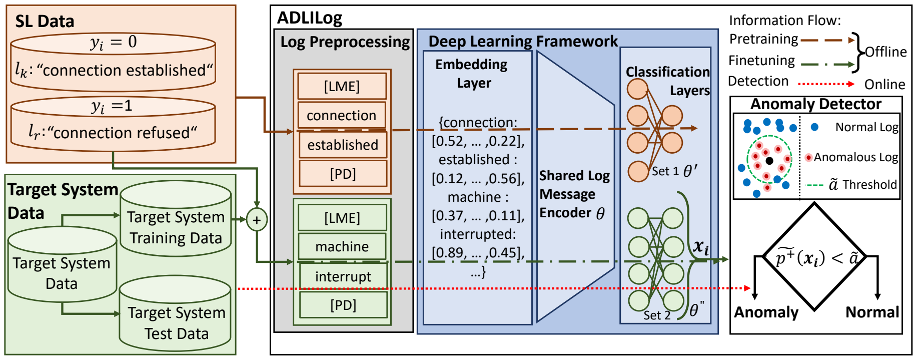

Fig. 1. ADLILog: Detailed design of the log anomaly detection method

during pretraining. The *second set* of classification layers, with parameters $\theta^"$, has two linear layers (*Set 2* in Fig. 1). The two layers have the same number of neurons equal to the *model size*. The output of the second set of linear layers is given as input for the loss function during finetuning. Additionally, the output of this layer is used as the log representation and is given as input to the anomaly detector.

4) Learning Process: The learning process is split into two phases: pretraining and finetuning. During the **pretraining** phase, we update the parameters of the embedding layer, the log message encoder and the first set of classification layers. We perform the pretraining with the SL data, using the binary cross-entropy [16]. After pretraining, the parameters of the encoder and the embedding layer (as *pretrained model*) are frozen and no longer updated. The pretrained model is used as a feature extractor for finetuning.

For the **finetuning** phase, we pair the pretrained model with the second set of linear layers. The training data consists of the target-system data and the "abnormal" severity group from SL. As we assume that the majority of the target-system data is normal (i.e., class 0), and by considering the "abnormal" class (i.e., class 1) as positive, the finetuning is addressed as binary classification. The "abnormal" class is always available, thereby, ADLILog does not need labeled target-system data. In the fine-tuning phase, we update just the parameters of the second set of linear layers ($\theta^"$). The finetuning enables learning the specifics of the target-system data while relying on the anomaly-related information from the SL data.

The finetuning loss determines the form of the final log vector embeddings. As a binary classification problem, multiple loss choices for finetuning are possible (e.g., binary cross-entropy [16], or hyperspherical loss [17]). The literature on anomaly detection [17] suggests that preserving the *concentration* property as enabled by the hyperspherical loss when learning representations can often improve the performance. Consequently, we use it for finetuning. Eq. 1 defines it.

$$L^i_{ad} = (1-y_i)||g(\mathbf{x_i};\theta,\theta^"))||^2 - y_i log(1-exp(-||g(\mathbf{x_i};\theta,\theta^"))||^2)) \quad (1)$$

where $\mathbf{x_i}$ is the log representation as output from the neural network, $y_i \in \{0,1\}$ is a label for the target-system data or the "abnormal" SL data, θ and $\theta^"$ are the fintuning network parameters, and $g(\mathbf{x_i};\theta,\theta^")$ is the learned function.

C. Anomaly Detector

Given the learned log representation ($\mathbf{x_i}$), the anomaly detector highlights the anomalous target-system logs. It has two components, i.e., 1) an assumed normality function $\tilde{p}^+_{ad}$, and 2) anomaly decision rule. The normality function is an assumed model of the normal target-system logs. The form of the function depends on the type of finetuning loss. The hyperspherical loss learns a model that places the normal logs (class 0) close to the centre of the hypersphere. Therefore, the smaller distances correspond to normal system behaviour and vice versa. Consequently, we use the reciprocal value of the Euclidean distance between the log representation $\mathbf{x_i}$ and the hypersphere centre, given by Eq. 2 as a normality function.

$$\tilde{p}^+_{ad}(\mathbf{x_i}) = \frac{1}{||\mathbf{x_i}-\mathbf{c}||^2}, \qquad \mathbf{c} = \mathbf{0} \quad (2)$$

To detect anomalies, we apply a decision rule on top of the normality function score. The decision rule sets a decision threshold $\tilde{a}$ over the scores, such that the logs with lower normality scores are reported as anomalous. The threshold $\tilde{a}$ is calculated on a separate validation set, composed of the normal target-system data and the "abnormal" SL class, such that a chosen performance criteria (e.g., F_1 score) is maximized.

IV. Experimental Design and Evaluation

A. Experimental Design

1) Datasets: BGL and HDFS are two benchmark datasets for log-based anomaly detection that are mostly used by the research community [5], [6], [9]. TABLE III shows the key datasets properties. Following He, et al. [5] we split the dataset into 80-20% train-test split. **HDFS** contains 11,175,629 logs generated from a map-reduce tasks on more than 200 Amazon's EC2 nodes [8]. Each log has a unique identifier (block_id) for each operation such as allocation, writing, replication and deletion. After parsing, there are 29 unique events, from which ten describe anomalous events and appear just when the block_id is anomalous. We used this observation to create the HDFS-sin dataset where each log line is labeled as anomalous or not. **BGL** contains 4,474,963 logs collected from a BlueGene/L supercomputer at Livermore Lab [18]. BGL has labels for individual log events given by the system administrators. We use these labels as ground truth information for single log line anomaly detection.

2) Competing Methods: We compare ADLILog with three state-of-the-art deep learning-based methods; two supervised (LogRobust and CNN) methods and one unsupervised (DeepLog) [6]. According to the log-based anomaly detection survey by Chen et al. [6], these three methods show the best detection performance on the two benchmark datasets. We used the public implementations of the methods available open-source in a GitHub repository [6]. The three methods were evaluated with the suggested values for their hyperparameters. Since the three methods require fixed input, similar to Chen et al. [6], we use $window\_size$ of 10 events to create fixed-size sequences and predict the next log (i.e., the stride is one). Following related work, we use three evaluation metrics (precision, recall and F_1) to estimate the detection performance of the compared methods [6].

TABLE III
DATASET PROPERTIES

Dataset	Time Span	# Logs	# Anomalies
HDFS	38.7 hours	11,175,629	16,838
BGL	7 months	4,747,963	348,460

3) ADLILog Experimental Setup: We evaluate ADLILog with the following hyperparameters: 1) *model size* $\{16, 64, 256\}$, 2) $max\_len = 32$ (it covers the majority of the log lengths) 3) *dropout* for regularization with $p = 0.05$, 4) *optimizer* Adam with $lr = 10^{-4}$ and $\beta_1 = 0.9$ and $\beta_2 = 0.99$, 5) *batch size* $= \{32, 64, 256, 512\}$.

B. Experimental Results and Discussion

For the **single log line anomaly detection comparison**, we compared ADLILog to the three state-of-the-art deep learning-based approaches. TABLE IV presents the results. ADLILog outperforms the supervised methods on almost all evaluation metrics. In particular, ADLILog showed the best predictive performance in terms of recall on both datasets (BGL-sin, and HDFS-sin). While DeepLog is being slightly better on recall on BGL-sin, it has significantly worsened performance on precision. On the F_1 as a primary evaluation metric, our method outperforms the supervised methods between 5-24% and the unsupervised one by 40-63%. The improvements of ADLILog are predominantly due to the rich set of abnormal events from the many diverse log instructions that help to discriminate the anomalous logs. ADLILog has a significant practical advantage in comparison to the competing supervised approaches because *does not require labeled target-system anomalies*. Therefore, it can be directly applied to a target system while obtaining detection performance similar or even better than the supervised approaches (which will still require expensive manual labeling). Therefore, the good performance of ADLILog comes at a smaller practical cost.

TABLE IV
SINGLE LINE LOG ANOMALY DETECTION COMPARISON

Single Log Line window size: 10 stride: 1	BGL-sin			HDFS-sin		
Method	F1	Prec.	Recall	F1	Prec.	Recall
ADLILog	**0.61**	0.55	0.70	**0.98**	1.00	0.96
DeepLog	0.21	0.12	0.82	0.35	0.62	0.24
LogRobust	0.37	0.63	0.26	0.89	1.00	0.79
CNN	0.56	0.47	0.68	0.88	1.00	0.78

Further, we noted that the predictive performances of all the methods for the BGL-sin dataset are significantly lower compared to the HDFS-sin. For example, the F_1 score of LogRobust from 0.89 on HDFS-sin falls to 0.37 on BGL-sin. One reason for this is that the logs from BGL-sin are from different simultaneously running tasks. There is a large difference between the nearby logs, a phenomenon in log analysis known as unstable sequences [9]. In BGL-sin there are 26.94% new log events in the test data compared to the training data. Although some of the new events are normal, the methods that exploit the local context miss-detect them as anomalous (as seen by the drop in precision). DeepLog, as the state-of-the-art unsupervised method, leverages the local context to detect anomalies, and it is significantly affected by the unstable sequences. LogRobust and CNN leverage supervised information about the events, which helps to improve the performance. In contrast, the HDFS-sin dataset is characterized by high regularity in the sequences due to the data generation procedure. ADLILog is not affected by the local context differences as it examines each log independently.

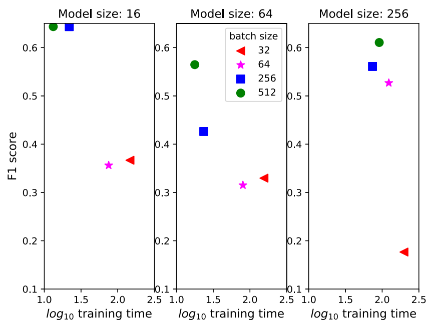

Fig. 2. Sensitivity analysis of the influence of batch and model size

The correct parameter setting influences the needed effort for fast configuration and the quality of the detection. To evaluate the **impact of the hyperparameters over the detection performance and efficiency**, we examined two hyperparameters, model and batch sizes, influencing the model performance and update time. We considered the BGL-sin dataset. The experimental results when varying the model size in the range $\{16, 64, 256\}$ and batch size in the range $\{32, 64, 256, 512\}$, reported in Fig. 2, show that the larger batch size and smaller model size provide better detection performances while being faster for updating. The prediction time per batch size of 512 is 17 ms ($\sim$30000 logs per second). Together with the small model size, these experiments imply that ADLILog has desirable practical properties.

V. Related Work

There exist multiple methods for the automation of log-based anomaly detection [7], [9], [10]. They are categorized into supervised and unsupervised. The supervised methods assume the existence of labels from the target system. There are several supervised deep-learning-based methods, e.g., LogRobust [9], and CNN [10]. LogRobust uses the LSTM architecture, augmented with attention. These two are popular deep learning architectures frequently combined for sequence modeling. LogRobust, as input, receives a sequence of events, and as output, it predicts if the observed sequence is anomalous or not. By careful sequence construction, i.e., by incremental sliding over the log sequences by one element, it can be used to predict single log lines [7]. Lu et al. [10] use Convolutions Neural Networks (CNN), another type of deep learning architecture, to learn normal and abnormal sequences from template indices. Supervised methods have strong detection performance, however, due to the large frequency of the software updates and large log volume the labeling process is expensive. Therefore, supervised methods are frequently considered impractical [5]. The unsupervised methods assume the absence of labels, having a practical strength. Xu et al. apply PCA [8] to learn the normal state of the event counts. The test samples are projected in the constructed vector space and reported as an anomaly if they significantly deviate from the learned normal state. DeepLog [7] is a popular unsupervised deep learning-based method. It introduces an auxiliary task called "next event prediction" (NEP). NEP is a supervised task that given a sequence of events, forecasts the most probable next event using an LSTM. The test sequences with an incorrect prediction for the next event are considered anomalous. As stated by the authors, DeepLog can be applied for sequential and single logs.

VI. Conclusion

This paper addresses the problem of automating log-based anomaly detection as a crucial maintenance task in enhancing the reliability of IT systems. It introduces a novel unsupervised method for log anomaly detection, named ADLILog. The key idea of ADLILog is to use the large unstructured information from the logging instructions of 1000+ GitHub public code projects to improve the target-system log representations, which directly improves anomaly detection. We first conducted a study to examine the language properties of the log instructions, and we show that they encode rich anomaly-related information. ADLILog combines the anomaly-related information and the target-system data to learn a deep neural network model by a sequential two-phase learning procedure. The extensive experimental results on the two most commonly used benchmark datasets show that ADLILog outperforms the related methods: the supervised by 5-24%, and the unsupervised by 40-63% on the F_1 score. Further experiments demonstrate that ADLILog has desirable practical properties concerning the time-efficient model updates and small model sizes. This study signifies the benefit of using large unstructured information in aiding the automation of IT operations.

References

[1] P. Notaro, J. Cardoso, and M. Gerndt, "A survey of aiops methods for failure management," *ACM Trans. Intell. Syst. Technol.*, vol. 12, 2021.

[2] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan, "A qualitative study of the benefits and costs of logging from developers' perspectives," *IEEE Transactions on Software Engineering*, pp. 1–17, 2020.

[3] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th ICSE*. New York, NY, USA: Association for Computing Machinery, 2016, p. 102–111.

[4] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surv.*, vol. 54, 2021.

[5] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. New York, USA: IEEE, 2016, pp. 207–218.

[6] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *CoRR*, vol. 2107.05908, 2021.

[7] 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*. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298.

[8] 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 22nd SOSP*. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–132.

[9] X. Zhang and et. al., "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on ESEC/FSE*. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–817.

[10] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *IEEE 16th Conf on Dependable, Autonomic and Secure Computing*, 2018, pp. 151–158.

[11] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd. (2020) spaCy: Industrial-strength Natural Language Processing in Python. Explosion.ai. [Online]. Available: https://doi.org/10.5281/zenodo.1212303

[12] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018, p. 178–189.

[13] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. USA: Wiley-Interscience, 2006.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates, 2017, p. 6000–6010.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[17] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 756–795, 2021.

[18] 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*. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 575–584.