How to Configure Masked Event Anomaly Detection on Software Logs?

Jesse Nyyssölä
M3S, ITEE, University of Oulu
Oulu, Finland
jesse.nyyssola@oulu.fi

Mika Mäntylä
M3S, ITEE, University of Oulu
Oulu, Finland
mika.mantyla@oulu.fi

Martín Varela
Profilence
Oulu, Finland
martin.varela@profilence.com

***Abstract*—Software Log anomaly event detection with masked event prediction has various technical approaches with countless configurations and parameters. Our objective is to provide a baseline of settings for similar studies in the future. The models we use are the N-Gram model, which is a classic approach in the field of natural language processing (NLP), and two deep learning (DL) models long short-term memory (LSTM) and convolutional neural network (CNN). For datasets we used four datasets Profilence, BlueGene/L (BGL), Hadoop Distributed File System (HDFS) and Hadoop. Other settings are the size of the sliding window which determines how many surrounding events we are using to predict a given event, mask position (the position within the window we are predicting), the usage of only unique sequences, and the portion of data that is used for training. The results show clear indications of settings that can be generalized across datasets. The performance of the DL models does not deteriorate as the window size increases while the N-Gram model shows worse performance with large window sizes on the BGL and Profilence datasets. Despite the popularity of Next Event Prediction, the results show that in this context it is better not to predict events at the edges of the subsequence, i.e., first or last event, with the best result coming from predicting the fourth event when the window size is five. Regarding the amount of data used for training, the results show differences across datasets and models. For example, the N-Gram model appears to be more sensitive toward the lack of data than the DL models. Overall, for similar experimental setups we suggest the following general baseline: Window size 10, mask position second to last, do not filter out non-unique sequences, and use a half of the total data for training.**

***Index Terms*—software execution logs, anomaly detection, software testing, deep-learning, LSTM, CNN, N-gram, failure localization, Masked Event Prediction, software debugging**

I. Introduction

Software system logs are considered as one of the primary sources for determining the cause of failure [1]. Log event prediction is one of the strategies of anomaly detection providing means for root cause detection as well as failure identification, tolerance, and recovery [1]. There is existing work that has focused on event prediction for anomaly detection e.g., [2]–[4]. Pinpointing anomalous events can help test and application engineers to identify and fix subtle bugs that result in crashes after a long time [2].

Existing work shows that event prediction can provide scores for log events of a software log under investigation [2]. These scores represent event suspiciousness or anomalousness and may help software engineers that are going through a log file that can be tens of thousands of lines long. Data for event prediction comes from historical logs that are free from anomalies. Such logs can be collected from durations when operations are known to be normal or from test runs that have ended without failures. As in prior work, we reason that incorrectness or low scores in event predictions in anomalous sequences are good candidates for true anomalous events. This paper extends prior work [2] by investigating more models, datasets, and settings. For example, this study introduces CNN as another DL model next to LSTM as it has proven to be an effective alternative for anomaly detection [5], [6].

As there are many configurations for the task, the key contribution of this paper is to provide a baseline for similar experimental setups for the future. This paper will proceed to introduce related work in Section II. Followed by that is research methods in Section III that explains the experimental environment, setup and practices. The results chapter (Section IV) will showcase the findings along with discussion on their relevance. After that suggested baseline is presented in Section IV-E and the paper concluded in Section VI.

II. Related work

A recent systematic literature review [1] analyses the qualitative and performance metrics of datasets, technical approaches and automated tools with an emphasis on failure detection and prediction. The review presents results from several studies that have utilized the same DL models and same public datasets used in this study. However, NLP is only discussed in the context of preprocessing and providing vectors for DL models. For example, Wang et al. [7] have demonstrated the effect of NLP based feature extraction in combination with LSTM.

Mäntylä et al. [2] utilized LSTM and N-Gram models to pinpoint anomalous events in log files. The study focused on the company Profilence that provides test automation and telemetry solutions. Their log files result from long-term stability tests, which generate very large amounts of log data. The results showed that using a light-weight approach (N-Grams) works as almost as well as more complex approaches such as deep learning, at a fraction of the computational cost.

Bogatinovski et al. [3], utilized masked event prediction and LSTM to introduce self-supervised method for detecting

anomalies in distributed traces. The results show high performance on experimental testbed data. However, the study only considered a handful of configurations on a single dataset. Our study aims to examine the impact of individual settings on multiple datasets.

Outside of anomaly detection, the success of neural language models has been proven over statistical models for masked word prediction [8]. For natural language, massive data sets exist that make it possible to train massive transformer-based models. Whereas for software logs one is often able to utilize only previous logs from the same system.

Kim [9] demonstrated the usage of CNN for natural language, building a CNN on top of the publicly available word2vec vectors, and showing good results with a single convolution layer. Lu et al. [5] used a similar method to compare CNN and LSTM models on the HDFS dataset for anomaly detection. Their results show that CNN can reach a higher and faster detection accuracy than LSTM. Chen et al. [6] provide a comprehensive analysis on deep and machine learning techniques that includes LSTM and CNN. They use HDFS and BGL as datasets for the study, and their results also show that CNN is generally better than LSTM. However, on the HDFS dataset LSTM reaches higher precision than CNN [6]. These studies show the efficacy of both LSTM and CNN for anomaly detection but do not consider the masked event prediction approach.

III. Research methods

This chapter outlines the research methods of the study. Further details on the implementation can be found via the replication package[1].

A. Experiment configurations

As this study aims to set the baseline for studying anomaly event detection, we investigate event prediction accuracy across various settings see Table I. We used four different datasets. The Profilence dataset [2] corresponds to log traces of about 280 runs of a test case related to an Android camera app (totaling ∼780MB of text), among which there are some failures. The HDFS dataset [10] consists of log files from a distributed system including over half a million labelled sequences. The BGL [11] dataset was produced by a BlueGene/L supercomputer resulting to over 4.7 million messages. The log contains alert category tags to indicate alert and non-alert messages [12]. The Hadoop dataset is based on two applications running on an underlying Hadoop platform generating logs [13]. It is the smallest dataset in this study with under 50MB of data. As these datasets differ in fundamental ways, they can show if some of the other settings only influence specific kind of data.

We use three models in the study: CNN, LSTM and N-Gram. CNN and LSTM are deep learning models while the N-Gram model is based purely on probabilistic assessment of n-grams. All the other settings will be run on these three models and four datasets, which means there will be 12 results for each of the other settings.

TABLE I: Settings for experiments

Setting name	Default	Studied configuration options
Data	All	Profilence, HDFS, BGL, Hadoop
Model	All	CNN, LSTM, N-Gram,
Sliding window	5	2, 5, 10, 15, 20
Mask position	0	0, 1, 2, 3, 4
Data selection	Total data	Total data, Unique data
Split for train data	0.5	0.1, 0.25, 0.5, 0.75, 0.9

The length of the *sliding window* determines how many surrounding events there are for a single prediction. The *mask position* setting then determines at which position of the sliding window the predicted event is. We note the mask position as the distance from the last event. For example, with window size 5 and mask position 0, the subsequence would look like $E_1\ E_2\ E_3\ E_4\ X$ where X represents the event to be predicted.

The settings *data selection* and *split* are both connected to training of the models. Split determines the portion of the whole dataset that goes to training while the rest is assigned as test data for inference. Using data selection, we can further reduce the training data into just unique sequences. Table II shows the number of sequences with an example of 50 percent split between training and test sets.

TABLE II: Number of sequences with 50 percent split

	Profilence	HDFS	BGL	Hadoop
Total normal sequences	100	558,223	866,675	169
Total training sequences	50	279,111	433,337	84
Unique training sequences	50	9,651	8,227	71

This study extends on the work of Mäntylä et al. [2] and will use the same default settings of window size 5, mask position 0, data selection total data and a 50-50 split. After gathering the initial results from each of the individual variables, we will combine them and see the results of the variables to be suggested as a baseline.

B. Computing Environment and DL model Configuration

All of the tests were run on a single computing environment provided by our anonymous HPC Cloud provider. Our virtual machine has Intel Core Processor (Broadwell, IBRS) with 28 cores at 2,4 Ghz. It has 224GB memory and two NVIDIA Tesla P100 PCIe 16GB graphical processing units.

Since we had over 200 experimental setups, we needed to have a limit on the training time of each of the DL models. We determined a dataset and model-specific epoch number that was based on a time budget. We trained each of the datasets with default settings on both of the DL models for five minutes which determined the number of epochs we used in other experiments, see Table III. Given our high-end GPU we

[1]https://github.com/M3SOulu/MaskedEventAnomalyDetection

think this time is sufficient to find results for our configuration space.

TABLE III: Number of epochs per model and dataset

	Profilence	HDFS	BGL	Hadoop
CNN	5	10	62	1,449
LSTM	5	8	50	1,145

As this study focuses on setting a generalised baseline for similar experimental setups, we did not go into dataset specific hyperparameter optimization although we recognize its value has been proven in NLP context as well [14]. Both of the DL models were implemented with Keras interface on top of TensorFlow libraries. The LSTM model uses embedding layer, two LSTM layers with 100 memory units, and two deep layers. The CNN model is otherwise the same, but rather than the LSTM layers, it uses a one-dimensional convolution layer and a global max pooling layer.

C. Pre-processing

All of the datasets consist of raw log lines that were parsed using the state-of-the art parser Drain [15] that has proven to be effective on various datasets [16]. As in work by Mäntylä et al. [2], we added start of sequence (SoS) and end of sequence (EoS) padding based on the sequence length and the position of the predicted event. This makes it possible to predict the first event and also the EoS event. In the experiments the data was split to training and test sets based on portions as shown in Table I.

IV. Results

The results of all test settings with the exception of data selection for unique data are summarised in Fig. 1. There are 12 box plots that correspond with a specific model-dataset pair. The plots illustrate well the similarity of the DL models as the CNN and LSTM plots are nearly identical while the N-Gram model falls short on the BGL and Profilence datasets (see more in Section IV-A). Another finding that becomes apparent on Fig. 1 is that some datasets are much more sensitive toward choosing the right settings than others. For example, the difference between the best and the worst settings (not including the setting for unique data only) for BGL CNN was 4.9 percent points while for Hadoop CNN it was 31.4 percent points.

This section continues by showcasing the results of the four setting variables individually. As there was no significant difference between LSTM and CNN results for window size, mask position or training data split, in those sections we refer to both of them as DL models.

A. Setting: Sliding window

Findings: Fig. 2a illustrates the results of N-Gram model for each dataset and the effect of varying the length of the sliding window. The main finding presented in the figure is the reduction in accuracy with Profilence and BGL datasets when window size was larger than five.

Fig. 2b shows effects of the window size on the DL models. While the results of the N-Gram model and the DL models are very similar up to n=5, after that the DL models can still keep the accuracy high or even improve it, while it starts to reduce on the N-Gram model. The DL models also work great on Hadoop that manages to improve the accuracy all the way to n=20, while on N-Gram it flattens out at 15.

Discussion: The findings are inline with previous work [2] that stated that the accuracy of the N-Gram model on Profilence data starts to reduce when window size is larger than 5 while this doesn't happen with LSTM model or the HDFS data at all. Based on this study we can extend the knowledge by noting that in this respect BGL data behaves exactly the same as the Profilence data while Hadoop resembles HDFS. To generalize the results, we recommend using the window size 10 as a baseline while also keeping in mind the model based differences.

B. Setting: Mask position

Findings: The effects on mask position are similar across all models (Fig. 2c and Fig. 2d). Regarding datasets, the findings indicate that the accuracy of predicting the very first and last event of the subsequence are the worst. While BGL is mostly unaffected by the mask position setting, all the other datasets show major improvement, when predicting the second to last position as opposed to the last one. The middle position is conflicting between datasets as Hadoop shows minor increase in accuracy while Profilence has minor and HDFS major reduction in accuracy.

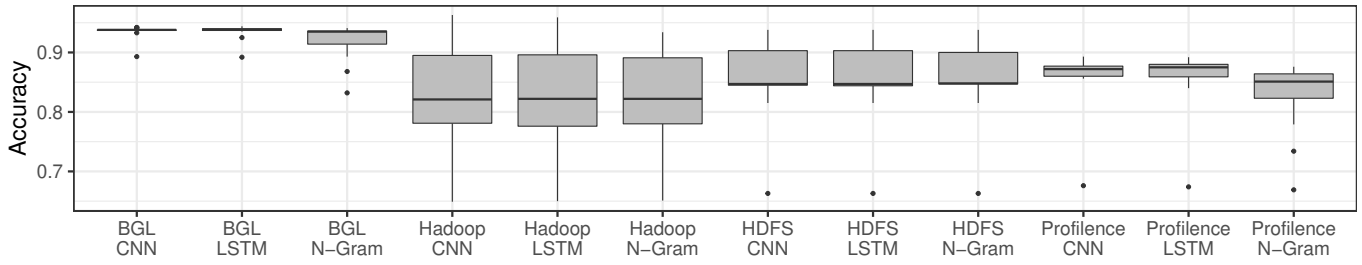
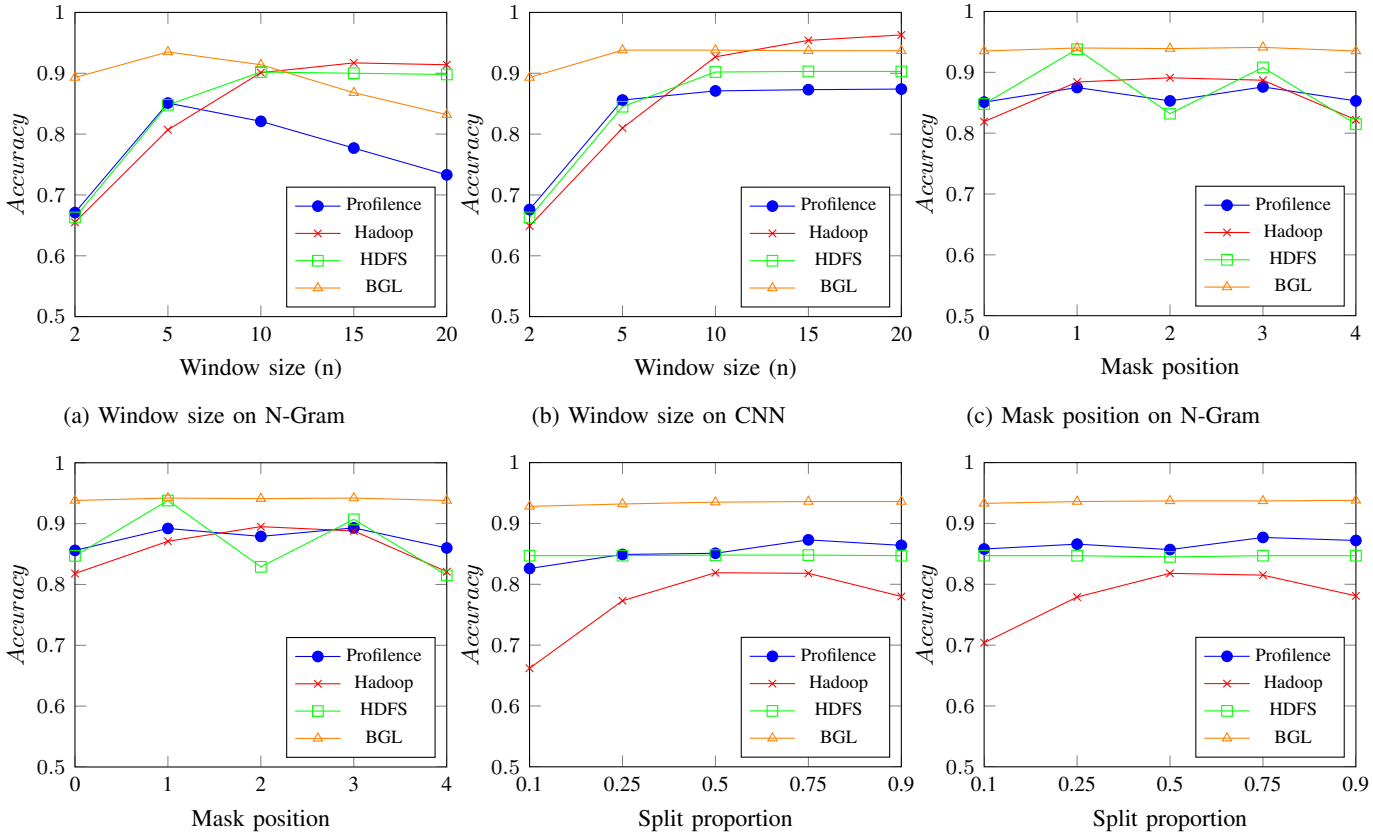


Fig. 1: Box plots summarising the results of configurations for each dataset and model.



(a) Window size on N-Gram (b) Window size on CNN (c) Mask position on N-Gram

(d) Mask position on CNN (e) Split for training data on N-Gram (f) Split for training data on CNN

Fig. 2: Charts showcasing the accuracy across various settings for N-Gram and CNN models

Discussion: The results show clearly that the mask positions 1 and 3 give the best performance with the position 1 being the best due to the large improvement in HDFS. The results are unable to explain the reduction in accuracy in the middle position of the subsequence. Future work should consider combining the longer window size with varying mask position to see whether the reduction in the middle is apparent with longer window size as well.

C. Setting: Split for training data

Findings: The effect of the split between training and test data varies significantly between datasets (Fig. 2e, Fig. 2f). HDFS and BGL have the most data and highest number of sequences which leads to consistent accuracy across all variations of the split proportion. While Profilence also has a relatively large amount of data, the low number of sequences leads to variation in the results as the splitting process introduces randomness for each split. As the smallest dataset, for Hadoop it is crucial to have enough data for training. For example, with the N-Gram model, increasing the proportion of training data from 0.1 to 0.5, increases the accuracy from 0.662 to 0.819. Increasing the proportion of training data higher than 0.5 showed a slight improvement for Profilence but a larger deterioration for Hadoop.

Discussion: Because having enough training data is mandatory for having a good model while also not seeing meaningful improvement using a higher split than 0.5, we suggest an even split between training and test sets as a baseline. However, we acknowledge that the split proportion is completely dependent on the amount of data available so we would recommend dataset specific split whenever possible. Reducing the training data can greatly reduce the training time as well while not having an effect on the accuracy.

D. Setting: Unique training data

Findings: Table IV shows the results of utilizing unique training data with each dataset and model. Because all of the Profilence sequences are unique, there was naturally no difference in the results. For the most part, the same is true for Hadoop as the there were only 13 duplicate sequences. When using only unique sequences in training, HDFS and BGL experience a major drop in the number of available sequences for training. This causes the accuracy to reduce on each test. However, for LSTM and N-Gram on BGL we find a massive drop as the accuracy goes from over 93 percent accuracy to 25.1 and 22.3 percent respectively.

Discussion: Given that the selection of only unique data for training can ruin the performance on some datasets and models (i.e., N-Gram or LSTM on BGL), we would not recommend

TABLE IV: Total vs. unique training data accuracy

	BGL		Hadoop		HDFS		Profilence	
	Total	Unique	Total	Unique	Total	Unique	Total	Unique
LSTM	**0.938**	0.251	0.817	0.817	**0.847**	0.768	0.856	0.856
CNN	**0.938**	0.884	0.81	**0.812**	**0.845**	0.794	0.856	0.856
N-Gram	**0.935**	0.223	**0.819**	0.818	**0.848**	0.824	0.851	0.851

using it as a baseline. However, since the training time is proportional to the size of the training data, we acknowledge that with some datasets training with unique data can lead to great improvements in computational efficiency.

E. Implications - Suggested baseline

The investigation into individual settings leads to the following variables to be suggested as the baseline: Window size 10, mask position second to last, do not filter out non-unique sequences, and use a half of the total data for training. Compared to the default settings used in this study, the window size increases and the mask position changes.

TABLE V: Default settings vs. suggested baseline

	BGL		Hadoop		HDFS		Profilence	
	Default	Baseline	Default	Baseline	Default	Baseline	Default	Baseline
LSTM	0.938	**0.946**	0.817	**0.953**	0.847	**0.955**	0.856	**0.919**
CNN	0.938	**0.943**	0.81	**0.953**	0.845	**0.954**	0.856	**0.913**
N-Gram	**0.935**	0.918	0.819	**0.938**	0.848	**0.954**	**0.851**	0.833

Finally, to get concrete results of the new baseline, we ran the models once more. Table V showcases the result of the suggested baseline compared to the default settings used in this study. While there are minor reductions in accuracy with the N-Gram model on BGL and Profilence, there are improvements across the board elsewhere. For example, using the baseline settings with the DL models on Hadoop increased the accuracy by approximately 14 percent points.

V. Limitations

In this study, there were some limitations caused by the inherent nature of the DL models. Training the models on a time budget means that the models were not necessarily fully trained. This could have impacted the results, but it also represents some realistic use cases (e.g., in one of our industrial cases, this is being incorporated into a user-facing product, and hence time is an important factor). Also, due to the large number of variations and long training time, we could not get more samples for results where we knew randomness would play a role. These are, for example, splitting Profilence data or choosing unique data from BGL.

Regarding the DL models, we decided not to engage in model specific hyperparameter optimization as it would have introduced additional parameters on top of the experimental settings. Hyperparameter optimization for masked event prediction should be done as part of future work.

VI. Conclusion

This study set out to provide a generalized baseline for settings when detecting anomalous events through masked event prediction on software logs. The settings included window size, mask position, unique data selection and proportion of the split for training data. All of the settings were ran on three different models and four different datasets. Based on our results, we could suggest a baseline that provided significantly better results than our initial default values (Table V).

References

[1] D. A. Bhanage, A. V. Pawar and K. Kotecha, "IT Infrastructure Anomaly Detection and Failure Handling: A Systematic Literature Review Focusing on Datasets, Log Preprocessing, Machine & Deep Learning Approaches and Automated Tool," in IEEE Access, vol. 9, pp. 156392-156421, 2021, doi: 10.1109/ACCESS.2021.3128283.

[2] M. Mäntylä, M. Varela, and S. Hashemi, "Pinpointing Anomaly Events in Logs from Stability Testing–N-Grams vs. Deep-Learning," *arXiv preprint arXiv:2202.09214*, 2022.

[3] J. Bogatinovski, S. Nedelkoski, J. Cardoso and O. Kao, "Self-Supervised Anomaly Detection from Distributed Traces," *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020, pp. 342-347, doi: 10.1109/UCC48980.2020.00054.

[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*, pp. 1285–1298, 2017

[5] S. Lu, X. Wei, Y. Li and L. Wang, "Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network," *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 151-158, doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037.

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

[7] M. Wang, L. Xu and L. Guo, "Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning," *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*, 2018, pp. 140-144, doi: 10.1109/ICFSP.2018.8552075.

[8] F. Atlınar, T. Ayar, A. Darrige, S. AlQays, A. Bağcı and M. F. Amasyali, "Masked Word Prediction with Statistical and Neural Language Models," *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2020, pp. 1-4, doi: 10.1109/ASYU50717.2020.9259862.

[9] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[10] W. Xu, L. Huang, A. Fox, D. Patterson, and M.I. Jordon. "Detecting large-scale system problems by mining console logs," in *SOSP'09: Proc. of the ACM Symposium on Operating Systems Principles*, 2009.

[11] A. Oliner and J. Stearley, "What Supercomputers Say: A Study of Five System Logs," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 575-584, doi: 10.1109/DSN.2007.103.

[12] S. He, J. Zhu, P. He, and M. R. Lyu. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. Arxiv, 2020.

[13] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang and X. Chen, "Log Clustering Based Problem Identification for Online Service Systems," *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 2016, pp. 102-111.

[14] A. Aghaebrahimian, and M. Cieliebak. "Hyperparameter tuning for deep learning in natural language processing." *4th swiss text analytics conference (swisstext 2019), winterthur, june 18-19 2019*. Swisstext, 2019.

[15] 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)*, pp. 33–40, 2017.

[16] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu. "Tools and Benchmarks for Automated Log Parsing". *International Conference on Software Engineering (ICSE)*, 2019.