Optimizing Anomaly Detection in Large-scale Logs

Arnatchai Techaviseschai
Computer Engineering Department
King Mongkut's University of Technology Thonburi
Bangkok, Thailand
arnatchai.tech@kmutt.ac.th

Sansiri Tarnpradab
Computer Engineering Department
King Mongkut's University of Technology Thonburi
Bangkok, Thailand
sansiri.tarn@kmutt.ac.th

Vasco Chibante Barroso
European Organization for Nuclear Research (CERN)
Geneva, Switzerland
vasco.chibante.barroso@cern.ch

Phond Phunchongharn
Computer Engineering Department
King Mongkut's University of Technology Thonburi
Bangkok, Thailand
phond.p@mail.kmutt.ac.th

***Abstract*—The ALICE detector at the CERN LHC is a large and complex system that has employed system logging to keep track of progress and detect any abnormal activity that may occur. In this study, we propose a complete log-anomaly detection framework to automatically detect anomalies, with an emphasis on examining its scalability when applied to large datasets as those typical of large high-energy physics experiments. Furthermore, we investigate different factors that may have an impact on model performance through extensive tests on real-world datasets, HDFS, and CERN Infologger. The insights gained from this study will enhance the safety, efficiency, and reliability of infrastructure operations at the ALICE facility.**

***Index Terms*—ALICE, CERN, Anomaly Detection, Monitoring system, Machine Learning, Convolutional Neural Network (CNN)**

I. Introduction

CERN (The European Organization for Nuclear Research) is the world's largest organization for research in particle physics. Over the years, significant experiments have been carried out to exploit particle collisions at various accelerator complexes and address a wide range of physics topics. Provided that each experimental run is cost and time-intensive, it is essential to monitor the detector-response systems, both manually (i.e. human monitoring) and automatically (i.e. system logging), and take prompt action if any issue emerges.

Log is commonly used for both system monitoring and troubleshooting since it contains information about the system runtime and activities. For large and complex systems, such as CERN's detectors, the amount of log messages generated per experimental run is considerable, especially during a system upgrade in which several modifications made to the software are done. Due to this, an automatic approach to troubleshooting or detecting system anomalies through log messages, so-called *log anomaly detection*, is thus desirable.

In particular, log anomaly detection is a technique used to discover unusual events in system log messages. Numerous approaches have been proposed to detect log anomalies, from traditional methods to recent advancements based on deep learning. Nevertheless, deep learning has demonstrated great potential in capturing intricate patterns and relationships within sequential data, making it suitable for log analysis and detection.

In this study, we present a deep learning framework based on Convolutional Neural Networks for detecting anomalous activity in log data generated by ALICE – the CERN experiment that primarily focuses on exploring the physics of interacting matter at extreme energy densities. We put emphasis on investigating the scalability of the model when provided with a large amount of data, and exploring the factors that contribute to performance improvement. This study made the following contributions:

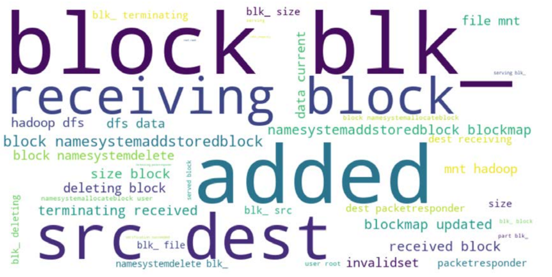
- We present a complete framework for automatic log anomaly detection, with a focus on examining its scalability when applied to large datasets. The insights gained will be particularly beneficial in improving the safety, efficiency, and reliability of critical infrastructure operations at the ALICE facility.
- We extensively explored different hyperparameters that are crucial in improving the model performance and demonstrated their effects on the model.
- Extensive experiments were conducted on two real-world datasets, HDFS and CERN Infologger. Results indicated that size of dataset has a significant impact on the hyperparameter settings.

II. Log Data and Analysis

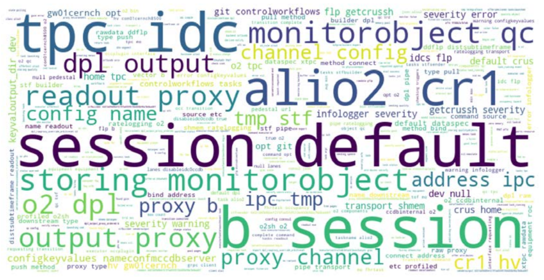
Two datasets are used in this study, namely HDFS and Infologger. HDFS[1] is a open-source data used by many research works for algorithm testing [1, 2] . Yet, HDFS data represent a less complex system compared to those from large data centers. Infologger, on the other hand, is a large complex dataset from ALICE O^2(Online-Offline) logging system in CERN, collected across processes running on multiple machines in each experimental run. The size of HDFS and Infologer are 2.2 GB and 10.7 GB, respectively. More details and examples of both datasets are presented in TABLE I and II.

[1]https://zenodo.org/record/3227177

Given that both HDFS and Infologger have the same property, namely usage of session logs, a session is thus used in this study as the unit to determine whether there exists an anomaly. In other words, an entire session will be labeled as an anomaly session when an abnormal event occurs. We generated session logs by grouping the PID of log messages since PID is a common feature of both datasets, as shown in TABLE II. Furthermore, it can be noticed from the table that, although the Infologger dataset has less sessions, the length of its log messages on average is very large. In terms of complexity and word variety, the Infologger dataset is more complex and contains a dense bag of words compared to HDFS, as depicted in Fig. 1.



(a) HDFS word cloud



(b) Infologger word cloud

Fig. 1. Dataset word cloud

III. Log Anomaly Framework

This section presents the log anomaly framework, comprising data preprocessing steps (i.e. log parsing, text encoding, session windowing), and the proposed CNN-based model.

A. Log parsing

Log parsing is the process of analyzing and extracting relevant information from log files. For anomaly detection, log parsing alleviates the number of log messages by grouping them into templates, thereby reducing the number of patterns the model needs to learn as well as making the model generalize better. This study utilized Drain [3] to discover and extract templates from log messages. For instance, based on the content column in TABLE I, a corresponding template generated from Drain will contain a unique template id, template, along with a separate list of parameters.

B. Text encoding

Text encoding transforms raw textual data into numerical representations. EventTemplate in a parsed log (e.g. id <*> client disconnected) is used as input. Then, data cleansing and tokenization are applied to remove unnecessary information (e.g. punctuation). We then created a dictionary from tokens, where each token is indexed by its occurrence frequency.

C. Session windowing

After text encoding, session windowing is applied to input data streams where the notion of sessions is relevant. It is a technique used to group related events or data points within a specific timeframe. Each window is defined with the process id (PID); therefore, any event falling within the specified PID is considered part of the same session. We note that log messages are further cropped to ensure that they are all of the same window size.

D. Proposed CNN-based Model

When applying CNNs to log anomaly detection, the log data is typically treated as a time window, where each log entry represents a data point in the sequence.

From Fig. 2, the incoming log keys sequence will feed through the input layer. It keeps the log key sequence in a specific shape before passing it to the embedding layer, transforming into vector representation in latent space. Next, stacked 1-dimensional CNN layers are applied using different kernel sizes to capture various patterns and contexts from log key sequences. The following layer is a Rectified Linear Unit (ReLU), which is used as an activation function, followed by max pooling to extract most representative features. Those features are then merged via a concatenation layer, flattened, and finally fed to the dense layer for classification. Softmax activation is applied to compute probability scores indicating whether the log key is normal or an anomaly.

TABLE I
Example of Dataset

HDFS dataset

LineId	Date	Time	Pid	Level	Component	Content
1	081109	203518	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010

Infologger dataset

LineId	timestamp	pid	hostname	severity	system	username	facility	Content
1	1617193614.448	2298	alio2-cr1-hv-head01	I	GUI	gui	Framework	id 0 client disconnected

TABLE II
DATASET DESCRIPTION (TOP: HDFS, BOTTOM: INFOLOGGER)

Name	**Description**
Pid	process id of application
Level	severity (i.e. INFO, DEBUG, WARN, ERROR, FATAL)
Component	part of the system that generates logs
Content	log message information
pid	process id of application
hostname	host machine that run the application
severity	severity (i.e. I, D, W, E, F)
system	group of application (e.g. First Level Processor (FLP))
username	unused (currently, same as system)
facility	submodule or framework that runs under the system
Content	log message information

TABLE III
NORMAL AND ANOMALY PARTITION

Dataset	**Max # logs/sequence**	**Normal # sessions**	**Anomaly # sessions**	**Total # sessions**
HDFS	298	558,223 (97.07%)	16,838 (2.93%)	**575,061 (100%)**
Infologger	2,194,073	79,528 (99.52%)	387 (0.48%)	**79,915 (100%)**

IV. EXPERIMENTAL STUDY

This section describes details about the designed experiments. Our goal is to evaluate the effect of hyperparameters on the performance and the model scalability. Two datasets were used – HDFS is the general one and Infologger is the complex one. The datasets are split into training, validation, and testing with fractions 60%, 20%, and 20%, respectively.

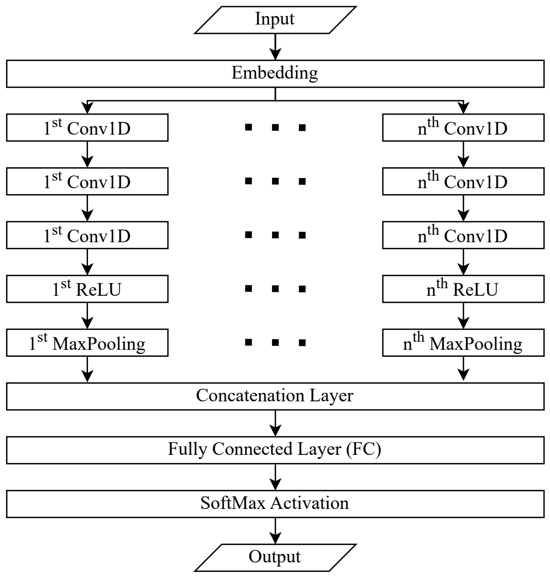


Fig. 2. Convolutional Neural Network (CNN) architecture

TABLE IV
HYPER-PARAMETERS SETTING OF EXPERIMENTS

No.	**Parameter**	**Values**
1.	Window Size	**<u>10</u>**, 20, 30, 40, 50
2.	Hidden Size	**<u>128</u>**, 256, 512, 768
3.	Embedding Dimension	**<u>64</u>**, 128, 256, 512
4.	Kernel Sizes	(1, 3, 5), **<u>(2, 3, 4)</u>**, (2, 4, 6), (3, 5, 7), (4, 6, 8), (5, 7, 9)
5.	No. of Convolutional layer	**<u>3</u>**
6.	Dropout	**<u>0.2</u>**
7.	Learning Rate	**<u>0.01</u>**

*Default values are in bold and underlined.

A. Hyperparameters

TABLE IV lists all hyperparameters of our experiments. Window size is the number of log messages that the model considers at a time when making predictions. Hidden size is the number of hidden units in the recurrent layer. Embedding dimension refers to the size of the embedding vectors of an input log message. Kernel sizes refer to sizes of the convolutional kernels used in the CNN model. Dropout is the probability of a neuron being dropped out during each iteration. Learning rate is the step size by which model's parameters are updated during each iteration.

For each experiment, only one hyperparameter is varied and others are fixed at their default settings. In this study, we vary window size, hidden size, embedding dimension, and kernel sizes. We discuss our observations for each of them in Section V.

B. Evaluation Metrics

Metrics used for evaluation include Accuracy, Precision, Recall, F1-score, False Negative Rate (FNR), and False Positive Rate (FPR). FNR is a misdetection rate, occurring when the model lets an abnormal log event slip through undetected. FPR is a false alarm, occurring when the model identifies normal data as anomalies. All formulas are shown in Eq. 1-6.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

$$FalsePositiveRate(FPR) = \frac{FP}{TN + FP} \tag{5}$$

$$FalseNegativeRate(FNR) = \frac{FN}{TP + FN} \tag{6}$$

where True Positive (TP) is the number of anomalous sessions correctly detected by the model. True Negative (TN) is the number of normal sessions correctly identified as normal. False Positive (FP) is the number of normal sessions

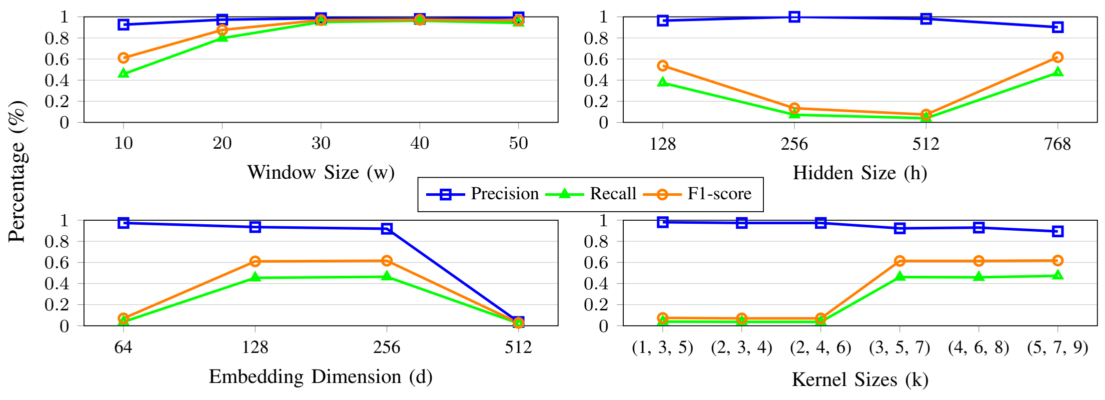


Fig. 3. Hyperparameter tuning of convolutional-based model on HDFS dataset

mistakenly detected as anomalies. False Negative (FN) is number of anomalous sessions misidentified as normal.

V. Results and Discussion

A. Hyperparameter effects for HDFS dataset

HDFS, a general dataset, could serve as a baseline for comparison with Infologger. In what follows, we discuss results obtained for each hyperparameter.

TABLE V
HDFS Experimental Results

Hyperparameters		Accuracy	Precision	Recall	F1	FNR	FPR
window_size	10	0.982	0.926	0.458	0.611	0.542	0.001
	20	0.988	0.973	0.798	0.876	0.203	0.001
	30	0.997	0.987	0.951	0.969	0.049	0.0006
	40	**0.998**	0.980	**0.963**	**0.972**	**0.037**	0.0006
	50	**0.998**	**0.995**	0.942	0.967	0.058	**0.0002**
hidden_size	128	0.980	0.964	0.374	0.537	0.627	0.0004
	256	0.971	**1**	0.072	0.134	0.928	**0**
	512	0.970	0.981	0.039	0.074	0.961	**0**
	768	**0.982**	0.902	**0.471**	**0.618**	**0.529**	0.002
embedding_dim	64	0.970	**0.974**	0.037	0.070	0.963	**0**
	128	**0.982**	0.935	0.454	0.610	0.547	0.001
	256	**0.982**	0.919	**0.464**	**0.616**	0.536	0.001
	512	0.970	0.036	0.023	0.027	**0**	**0**
kernel_sizes	1, 3, 5	0.970	**0.981**	0.039	0.074	0.961	**0**
	2, 3, 4	0.970	0.974	0.037	0.070	0.963	**0**
	2, 4, 6	0.970	0.974	0.037	0.070	0.963	**0**
	3, 5, 7	**0.982**	0.923	0.461	0.614	0.539	0.001
	4, 6, 8	**0.982**	0.930	0.459	0.614	0.541	0.001
	5, 7, 9	**0.982**	0.894	**0.473**	**0.618**	**0.527**	0.002

a) Window Size: Fig. 3 shows the performance improvement across all metrics when window size increases, reaching a peak at w=40 with F1-score of 0.972. We also observed that some metric starts to drop slightly when w=50, indicating that excessive features from input can start to harm the model by giving irrelevant information when increasing the window size.

b) Hidden Size: different hidden sizes allow the model to learn different representations and patterns in the data. At h=128, the model shows a fair result as some relevant information in an input sequence could be captured. However, increasing hidden size may not improve the model as it ignores anomalous patterns and focuses only on normal patterns. After h=512, the model has demonstrated to achieve the best F1-score of 0.618, which may signal that the model captures some interesting pattern or the presence of overfitting effects that could emerge in a small dataset. from a small dataset.

c) Embedding Dimension: Fig. 3 shows that moderate output values could be achieved at h=128 and 256, indicating that some specific dimension fits well with the dataset. We observed that the best F1-score of 0.616 was obtained for h=256. However, a lower dimension demonstrated to give poor results as it does not contain adequate information.

d) Kernel Sizes: The results show that the model cannot capture patterns with lower kernel sizes. This suggests that larger kernels are more suited for capturing broader patterns in input sequences. We also observed that recall and F1-score improve when kernel sizes increase in complexity. The best F1-score (0.618) was obtained for k=(5, 7, 9), emphasizing that wider filters can capture more patterns and thus, with a bigger picture, the model can better comprehend input sequences.

B. Hyperparameter effects for Infologger dataset

This section discusses the effect of hyperparameter variations based on Infologger, which is a larger and more complex dataset.

TABLE VI
Infologger Experimental Results

Hyperparameters		Accuracy	Precision	Recall	F1	FNR	FPR
window_size	10	**0.971**	**0.715**	0.831	**0.768**	0.170	**0.020**
	20	0.963	0.640	0.846	0.729	0.154	0.030
	30	0.965	0.657	0.863	0.746	0.137	0.029
	40	0.961	0.629	0.873	0.731	0.127	0.033
	50	0.961	0.633	**0.886**	0.738	**0.114**	0.034
hidden_size	128	0.417	0.089	0.9996	0.164	0.0004	0.619
	256	0.970	0.700	0.833	0.760	0.167	0.022
	512	0.970	0.703	0.834	0.762	0.166	0.021
	768	**0.974**	**0.740**	0.828	**0.781**	0.172	**0.018**
embedding_dim	64	**0.984**	**0.897**	0.811	**0.851**	0.189	**0.006**
	128	0.972	0.726	0.829	0.774	0.171	0.019
	256	0.970	0.692	**0.834**	0.756	**0.166**	0.023
	512	0.980	0.828	0.816	0.821	0.184	0.010
kernel_sizes	1, 3, 5	0.969	0.687	**0.832**	0.752	**0.168**	0.023
	2, 3, 4	**0.983**	**0.881**	0.812	**0.845**	0.188	**0.007**
	2, 4, 6	0.970	0.703	**0.832**	0.762	**0.168**	0.021
	3, 5, 7	0.965	0.650	0.831	0.729	0.169	0.027
	4, 6, 8	0.975	0.757	0.822	0.788	0.178	0.016
	5, 7, 9	0.972	0.716	**0.832**	0.769	0.169	0.020

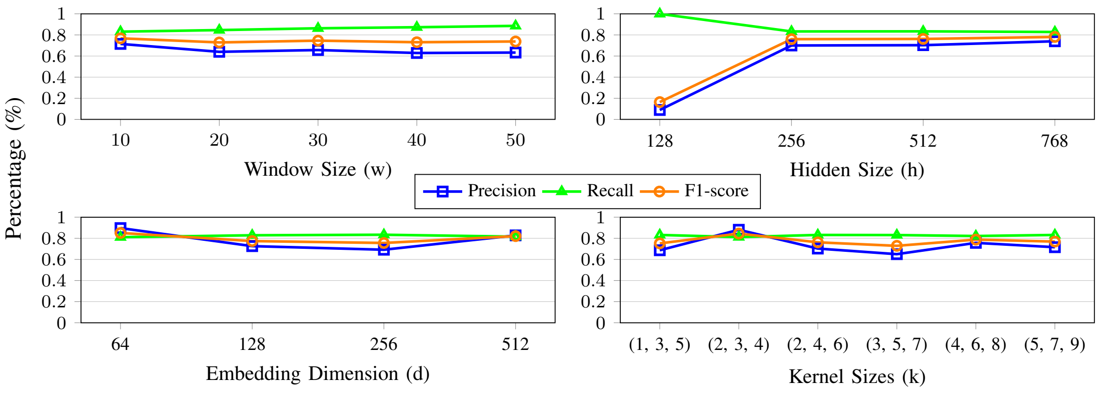


Fig. 4. Hyperparameter tuning of convolutional-based model on Infologger dataset

a) Window Size: Fig. 4 shows that F1-score and precision decrease as window size increases, indicating that too much context might introduce noises. Recall rises when the window size is larger, reflecting coverage of relevant log messages. Besides, TABLE VI shows that the highest F1-score (0.768) was achieved at w=10, while at w=50, FNR dropped to 0.114 because anomalies may span multiple log entries and a larger window helps the model to detect those patterns.

b) Hidden Size: a drastic change was observed at h=128, where F1-score and precision drop to almost zero in Fig. 4. This could be a sign of lacking input information when the hidden size is too small. It could also be noticed that starting from h=256, all metrics start to give better competitive results. At h=768, the model achieves the highest F1-score (0.781), demonstrating that a larger hidden size helps improving the model's performance, especially in terms of precision. With a smaller hidden size of h=256, FNR reduces at 0.167 which could be considered the best value, given that the case of h=128 yields the lowest accuracy.

c) Embedding Dimension: on average, higher recall and lower FNR were achieved with moderate embedding dimensions, i.e. h=128 and 256, indicating that an appropriate embedding dimension helps the model to focus only on key features of inputs. Nevertheless, the related F1-score and precision are lower as displayed in Fig. 4, indicating a tradeoff between model complexity and the amount of fine-grained information being captured. In addition, using d=64 achieves the highest F1-score (0.851) which is comparable to d=512.

d) Kernel Sizes: different kernel sizes capture different patterns in the log data. Fig. 4 shows that precision and F1-score are highest at k = (2, 3, 4), followed by k = (4, 6, 8) as it is double in size. The results reported in TABLE VI show that using kernel sizes 2, 3, and 4 leads to the best F1-score (0.845), indicating that these particular kernel sizes are effective in capturing important patterns in the log sequences.

C. Model performance across different datasets

The outcome indicates that the model is capable of handling large data relatively well, yielding linear trend results across all window sizes and all metrics. With a smaller dataset HDFS, the model also demonstrates the same trend for precision; however, its recall and F1-score clearly indicate that larger window sizes are more helpful for learning. To summarize, the model is limited to small window sizes for a small dataset HDFS, in contrast to Infologger where the trend appears to be increasing as the window size grows.

Additionally, the model yielded good results on HDFS with low and high numbers of hidden layers, whereas better results were obtained with large hidden size for Infologger. This emphasizes that size and complexity of the dataset directly affect hyperparameter settings. We further observed that a medium-sized embedding dimension is more suitable for HDFS, while low and high dimensions are more well-suited for Infologger. For the kernel sizes, the model yielded competitive results for Infologger in general. With HDFS, however, the model obtained good results at more complex kernel size patterns.

It is essential to consider specific requirements and constraints when selecting hyperparameters; minimizing FNR helps avoiding missing anomalies, while minimizing FPR helps reducing false alarms in the system. Since our goal is to capture anomalies, recall and FNR are the main indicators to identify when the model poorly performs. Finally, both datasets show that larger window sizes and hidden sizes generally lead to better performance, implying that the model can benefit from more context and learning capacity as the dataset size increases. This additionally suggests that the model has the potential to scale effectively to handle larger datasets.

VI. Related Works

Methods for log anomaly detection have progressed from statistical-based approach [4], traditional machine learning [5], to deep learning [6]. The proven success of deep learning has also led to an increase in various other text-based research and applications [7–10]. Unlike statistical methods, deep learning techniques like CNNs have shown superior capabilities in extracting intricate patterns and dependencies from sequential data. Numerous research studies have shown that CNN can capture relationships among events in system logs relatively well. The study by Lu et al. [11] is one of the first to apply CNN for anomaly detection in log events. The results from the proposed method reveal that the model could outperform several other competitive approaches. Likewise, a study reported in [12] proposed a comparative study of different architectures, the results of which showed that CNN could achieve the best outcome. Apart from CNN applications, RNN and its variation such as LSTM have also been used in several works [13, 14]. Nevertheless, hyperparameter settings for the experiment was not extensively studied nor discussed in these previous works. Regarding the log datasets, Loghub [1] provides a collection of system logs (e.g. BGL, Thunderbird, and HDFS) which are freely accessible for log analytics research.

VII. Conclusion

This study presents a framework for log anomaly detection using convolutional neural networks. By testing on different data sizes, our findings show that the scalability of CNN is possible for a large and complex dataset. The experimental results also give valuable insights into effects of different hyperparameters. In other words, by carefully selecting hyperparameter values while also taking into account tradeoffs in evaluation metrics, we can create a reliable model that practically works for different size and complexity of logging systems such as the ALICE O^2 facility at CERN.

Acknowledgement

This research was supported in part by National Science and Technology Development Agency (Project ID P-19-51650) and in part by the Big Data Experience Center, Thailand.

References

[1] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, *Loghub: A large collection of system log datasets towards automated log analytics*. arXiv: 2008.06448 [cs.SE].

[2] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP '09, Big Sky, Montana, USA: Association for Computing Machinery, 2009, 117–132. doi: 10.1145/1629575.1629587.

[3] 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)*, 2017, pp. 33–40. doi: 10.1109/ICWS.2017.13.

[4] S. He *et al.*, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018, Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, 60–70, isbn: 9781450355735. doi: 10.1145/3236024.3236083.

[5] 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 International Conference on Software Engineering Companion*, ser. ICSE '16, Austin, Texas: Association for Computing Machinery, 2016, 102–111, isbn: 9781450342056. doi: 10.1145/2889160.2889232.

[6] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.

[7] Q. Li *et al.*, "A survey on text classification: From traditional to deep learning," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 2, pp. 1–41, 2022.

[8] S. Tarnpradab, F. Liu, and K. A. Hua, "Toward extractive summarization of online forum discussions via hierarchical attention networks," in *The Thirtieth International Flairs Conference*, 2017.

[9] C. Ma, W. E. Zhang, M. Guo, H. Wang, and Q. Z. Sheng, "Multi-document summarization via deep learning techniques: A survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[10] S. Tarnpradab and K. A. Hua, "Attention based neural architecture for rumor detection with author context awareness," in *2018 Thirteenth International Conference on Digital Information Management (ICDIM)*, IEEE, 2018, pp. 82–87.

[11] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *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.

[12] P. Cheansunan and P. Phunchongharn, "Detecting anomalous events on distributed systems using convolutional neural networks," in *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*, 2019, pp. 1–5. doi: 10.1109/ICAwST.2019.8923357.

[13] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" In *Proceedings of the 44th international conference on software engineering*, 2022, pp. 1356–1367.

[14] X. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019, Tallinn, Estonia: Association for Computing Machinery, 2019, 807–817, isbn: 9781450355728. doi: 10.1145/3338906.3338931. [Online]. Available: https://doi.org/10.1145/3338906.3338931.