

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,400

Open access books available

174,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Chapter

Verification of Generalizability in Software Log Anomaly Detection Models

*Hironori Uchida, Keitaro Tominaga, Hideki Itai, Yujie Li
and Yoshihisa Nakatoh*

Abstract

With recent rapid technological advances, the automatic analysis of software logs has received particular attention. Currently, there is much research on the use of Deep Learning in the field of software log anomaly detection, and they have reported high accuracy of more than 0.9 in the f1-score. On the other hand, there are reports that it has not been used in the field of software development. We conducted a generalized evaluation against representative models for log anomaly detection to elucidate the cause of this problem. Five models were used in the subject: four representative models (two supervised and two unsupervised) and our proposed Neocortical Algorithm (supervised). We used the commonly used Blue Gene/L supercomputer log (BGL) dataset. The learning curves and cross-validation showed a tendency toward overfitting in all models. In addition, a survey of the frequency of log patterns confirmed the need for a more diverse dataset, as many of the patterns are a series of specific logs.

Keywords: anomaly detection, log analysis, log parsing, deep learning, software log, software development

1. Introduction

Automatic analysis of software logs has attracted significant attention due to the rapid development of technology in recent years. Currently, there are many studies with Deep Learning in the field of software log anomaly detection, reporting high accuracies surpassing 0.9 on the f1-score [1, 2]. On the other hand, it has been reported that Deep Learning for software log anomaly detection is not widely employed in the software development industry. The Loghub dataset, released by He et al., is presently frequently used in the field of software log anomaly analysis [3]. While Loghub contains logs from various systems, only one type of log is available for each system. Consequently, the accuracy of anomaly detection is assessed for only one pattern, and a comprehensive evaluation using multiple datasets is not performed. As a result, the effectiveness of the various anomaly detection models reported may be limited to specific datasets.

Therefore, to assess the generalizability of representative anomaly detection models across multiple dataset patterns, we initially conducted cross-validation using the Blue Gene/L supercomputer log (BGL) dataset from Loghub. For this purpose, we utilized the Deep-loglizer Toolkit developed by Chen et al. [4], which comprises four models, namely, CNN, LSTM (supervised learning), Transformer, and Auto Encoder (unsupervised learning). Additionally, we incorporated our proposed SPClassifier (supervised learning) to make use of a total of five models.

The second approach to evaluating generality involves using the validation datasets. In the study by Chen et al. that evaluates various models, the dataset is split into two partitions: the training dataset and the test dataset [1]. At each epoch, the models are evaluated on the test data, and the model with the highest accuracy in that epoch is considered the optimal model to calculate the accuracy for the test dataset. Given the potential for overfitting on the test dataset with this method, we split the dataset into three separate datasets for evaluation: the training dataset, the validation dataset, and the test dataset.

Furthermore, we examined the type and frequency of logs included in the dataset to assess whether the dataset is suitable for generic evaluation.

In summary, this experiment aims to clarify the following three points:

1. Evaluation of generality through cross-validation: Investigating the variation in accuracy due to differences in the types of logs included in the training and test datasets.
2. Evaluation of generality using the validation dataset: Assessing the generality using the validation dataset, which has not been included in previous benchmark studies.
3. Investigation of the log structure included in the dataset: Examining the similarity of the log structure in the commonly used BGL dataset to that used in software development.

2. Study design

This study uses the Toolkit (Deep-loglizer) provided by Chen et al. This Toolkit allows for flexibility in the model setup, including the ability to modify the loss function and determine whether or not to incorporate semantic information from the logs. We exclusively utilized sequential information in this experiment since our experimental setup lacked the necessary computational resources to handle semantic information. Notably, Chen et al. reported comparable accuracies with and without semantic information.

2.1 The common workflow

The common parts of the workflow for anomaly detection used in this experiment are shown in **Figure 1**. A standard anomaly detection model comprises four key steps: (1) Log Parsing, (2) Log Grouping, (3) Log Representation, and (4) Deep Learning Models [2]. Sections 2.1.1–2.1.4 provide an overview of each step and the specific techniques used in this study.

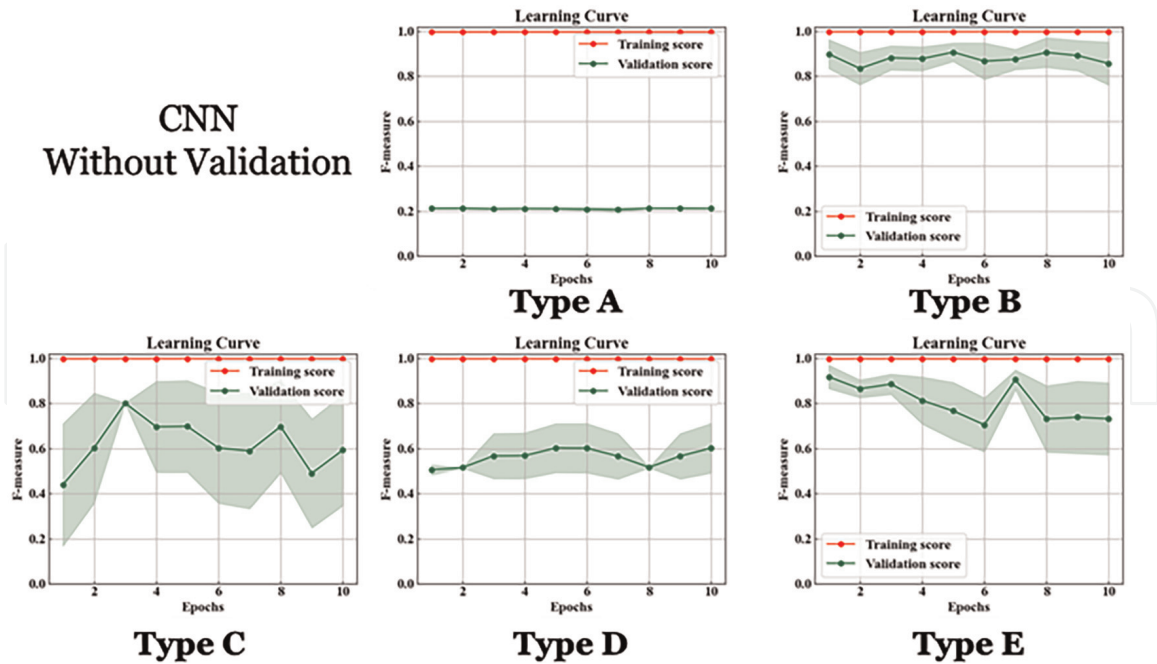


Figure 1.
 Learning curve for CNN.

2.1.1 Log parsing

The log contains different fields such as timestamp, process ID, and severity. However, for the line-by-line anomaly label dataset, only the crucial log message portion is extracted and used through log analysis. Log analysis is used to automatically translate each log message into a specific event template (constant part) associated with the parameters (variable part). Various log analysis techniques are available, such as frequent pattern mining, clustering, heuristics, etc. For this experiment, log analysis was conducted using Drain (heuristic) [5], which has been used in benchmark studies. Drain uses fixed-depth analysis trees to speed up the analysis process and encodes rules specifically designed for the analysis. The parameters used are as follows: Similarity threshold = 0.5, Depth of all leaf nodes.

2.1.2 Log grouping

This step separates the logs into various groups. Typically, three types of windows are employed for log grouping, namely, Fixed Window, Sliding Window, and Session Window. Fixed Window is a grouping technique that splits the logs according to their frequency of occurrence, whereas Sliding Window segments the logs into window size and step size. Session Window, on the other hand, leverages log identifiers to group logs with the same execution path. For the current study, the Sliding Window of 10 and Sliding Step of 1 are utilized for log grouping.

2.1.3 Log representation

After grouping the logs, the logs are represented in the various formats required by the DL model. In general, they are converted into (1) sequential vectors, (2) quantitative vectors, and (3) semantic vectors. (1) Sequential vectors reflect the order of log

events within a window. (2) Quantitative vectors use the frequency of occurrence of each log event within a log window. (3) Semantic vectors acquired from the language model represent the semantic feature of log events.

In this experiment, the (1) sequential vector is used.

2.1.4 Deep learning models

After the log representation step, the extracted features are fed into a deep-learning model for anomaly detection.

2.2 Evaluated models

In this experiment, we used a total of five models, consisting of two supervised learning models, namely Convolutional Neural Network (CNN) [6] and Long Short-Term Memory (LSTM) [7], and two unsupervised learning models, namely Auto Encoder (AE) [8] and Transformer [9], in addition to our proposed SPClassifier [10]. An overview of each model and the Toolkit parameters used in this study is provided in Section 2.2.1–2.2.5.

2.2.1 CNN (supervised model)

The input logs undergo the following preprocessing steps: first, they are converted to IDs, then to vectors using `logkey2vec`, and finally fed into the CNN model. This approach resulted in an impressive F-measure of 0.98 on the HDFS dataset. The specific Toolkit parameters used for this experiment are as follows: “`python cnn_demo.py -label_type anomaly -feature_type sequential - 10.`”

2.2.2 LSTM (supervised model)

This approach utilizes fixed-dimensional semantic vectors to represent log events and employs an attention-based Bidirectional Long Short-Term Memory (Bi-LSTM) classification model for anomaly detection. The Toolkit parameters used for this experiment are as follows: “`python lstm_demo.py -label_type anomaly -feature_type sequential -use_attention -topk 50 -epochs 10.`”

2.2.3 Auto encoder (unsupervised model)

The model consists of two Auto Encoders and one Isolation Forest; the Auto Encoder is used for feature extraction and anomaly detection, and the Isolation Forest is used for positive sample prediction. The parameters used in the Toolkit for this experiment are as follows: “`python ae_demo.py -feature_type sequential -anomaly_ratio 0.8 -epochs 10.`”

2.2.4 Transformer (unsupervised model)

Existing approaches exhibit limitations in their ability to generalize to new, unseen log samples. To address this issue, Logsy was proposed as a novel method for anomaly detection, utilizing a self-attention encoder network for hyperspherical classification. Logsy formulates the log anomaly detection problem by discriminating between

regular training data from the target system and samples from auxiliary log datasets that are easily accessible from other systems.

The parameters used in the Toolkit for this experiment are as follows: “python transformer_demo.py –label_type next_log –feature_type sequential –topk 50 –epochs 10 –use_attention.”

2.2.5 SPClassifier (supervised model)

SPClassifier is a model with sparse features and internal representations suitable for training in CPU environments [10]. The proposed method consists of one spatial pooling layer [11, 12] and one Feedforward Neural Network for classification, which identifies anomalous patterns from log data transformed into 2D features. The feature transformation process involves converting the input log sequence into a sparse distributed representation (SDR), which is a binary sequence of fixed dimensions. In the SDR, a specific percentage of the bits are set to 1, while the remaining bits are set to 0. These transformed features serve as input to the spatial pooling stage. Spatial pooling (SP) incorporates local suppression between adjacent mini-columns and implements a k-wins-take-all computation. At any given time, only a small fraction of the mini-columns with the most active inputs are active. Feed-forward connections to active cells are adjusted at each time step based on Hebb’s learning rule. Additionally, a homeostatic excitation control mechanism, referred to as “boosting,” operates on a slower time scale. Boosting enhances the relative excitability of underactive mini-columns, encouraging their activation and participation in the input representation. Subsequently, the transformed SDRs obtained through spatial pooling are fed into a classifier responsible for detecting anomalies. The employed classifier utilizes a single-layer neural network. It takes as input a flat binary SDRs array representing the output of the spatial pooling layer and predicts the abnormal or normal label. A softmax function is employed as the activation function for the output of the network. The network weights are updated during training using a formula based on the provided label information.

$$w_{ij} \leftarrow w_{ij} + \alpha \times \sum_{i=0}^{c-1} \left\{ \frac{1}{c} - \text{softmax} \left(\sum_{i=0}^{c-1} w_{ij} z_j \right) \right\} \quad (1)$$

where w_{ij} is the weight between the j -th value of the flattened input z_j and the i -th output node of the neural network. c is the number of categories, $c = 2$ since normal and abnormal categories are used for anomaly detection. α is a coefficient that controls the speed of learning.

The parameters are shown in **Table 1**.

Parameter	Value	Description
Window size	10	Size of sliding window
Stride	1	Step size of sliding window
Input SDR Length	500	Number of dimensions of SDR transformation for a single template index.
Input SDR Sparsity	0.15	Proportion of binary values equal to 1 across all dimensions post SDR conversion.

Parameter	Value	Description
Input Dimensions	(500, 10)	Shape of the coded image generated by stacking SDRs.
Column Dimension	(830, 15)	Shape of the columns in spatial pooling layer.
Potential Radius	7	Value that determines the input range over which one column has a potential connection.
Potential Pct	0.1	Percentage of inputs with potential connections in the hypercube.
Global Inhibition	True	Whether to consider all columns as neighbors when determining the active state of a column.
Local Area Density	0.1	Percentage of columns that can be active between neighbors.
Stimulus Threshold	6	Minimum number of synaptic connections required for a column to be active.
SynPermActiveInc	0.14	Amount of increase in permanence value of active synapses at each learning step.
SynPermInactiveDec	0.02	Amount by which the permanence value of inactive synapses decreases with each learning step.
SynPermConnected	0.5	Minimum permanence value at which a synapse is considered connected.
DutyCyclePeriod	1402	Length of time step considered when updating the boost factor based on how often each column is active.
MinPctOverDutyCycles	0.2	Lower limit on how often a column has active input above the stimulus threshold.
Boost Strength	7	Parameters that control the strength of the boost factor's adaptive effect.
WrapAround	False	Whether the first and last dimensions of the input are considered adjacent in the mapping between input and column.

Table 1.
Parameter list for SPClassifier.

2.3 Dataset

The BGL dataset collected from the supercomputer system Blue Gene/L was used as the evaluation dataset for this experiment. This dataset contains log data labeled with anomaly logs. It was sourced from Loghub [3], a renowned repository offering a vast collection of log datasets that can be employed for AI-based log analysis.

2.3.1 Data selection strategies

In this study, a set of time series was used, where 80% of the available logs were allocated for training and the remaining 20% for testing. The testing dataset was further split into two halves, with the initial half used for validation purposes and the second half used for testing. Such a partitioning strategy emulates a practical software development scenario, where past logs are utilized for training and future logs are used for testing purposes.

2.3.2 Different data grouping

There are two major data grouping techniques: window grouping and session grouping. In this experiment, we used the window grouping approach, utilizing a

window size of 10 with a sliding value of 1. It is worth noting that the results obtained using session windows outperform those obtained using fixed windows on the BGL dataset, as previously reported by Le et al.. This result may be attributed to the fact that the larger size of the input data augments the amount of information that can be acquired, which facilitates the detection of anomalies. In this study, we selected to use fixed grouping with a window size of 10, as we believe that anomaly detection at a more detailed level is important, particularly in a developmental setting.

2.4 Evaluation metrics

2.4.1 Evaluation method

In the accuracy comparison, each model after training is used to verify the accuracy of anomaly detection for test data. The classification performance of each model is evaluated by the F-measure value; F-measure is an evaluation index that indicates the balance between detection accuracy and the number of anomalies detected. Here, the F-measure is calculated as follows.

$$\textit{Precision} = \frac{TP}{TP + FP} \quad (2)$$

$$\textit{Recall} = \frac{TP}{TP + FN} \quad (3)$$

$$\textit{F - measure} = \frac{2 \cdot \textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}} \quad (4)$$

where:

TP represents abnormal instances correctly classified by the model.

TN represents normal instances correctly classified by the model.

FP represents normal instances misclassified by the model.

FN represents abnormal instances misclassified by the model.

2.4.2 Learning curve

Each training set consisted of one epoch, and during training the model was evaluated for accuracy using both training and validation data. The training range consists of epochs 1 through 10. In experiments without a validation set, the test data were evaluated as the validation set.

2.5 Objectives of the experiment

Objective 1: Generalizability evaluation with cross-validation.

To evaluate the generalizability of each model, we employed five-fold cross-validation in this experiment. Four-fifths of the dataset was used as the training dataset, while the remaining one-fifth was designated as the test dataset. The test dataset was subsequently split in half, with the first half serving as the validation dataset. By varying the split points and comparing the accuracy of anomaly detection, we will explore the potential impact of training dataset bias and test dataset bias on the performance of the system.

Objective 2: Generalizability evaluation with the validation dataset.

This experiment aims to evaluate the accuracy with and without the validation dataset, as previous studies only split the dataset into training and test data. The test data are used to determine the optimal model among multiple Epoch training runs; however, there is a potential risk of over-training on the test data. Hence, this study incorporates the validation dataset to assess whether over-training occurs. The dataset segmentation method was the same as Obj1, with train 80%, validation 10%, and test 10%. Our previous studies have shown that the accuracy is extremely high when the train dataset ratio is increased to 90% [13]. We attribute this result to the extreme reduction in the number of unknown anomaly logs not present in the training dataset, from 268 to 3, resulting in fewer opportunities to test the unknown anomaly logs. In the actual field of development, the source code is updated daily and the logs are updated accordingly. Therefore, we use 80% of the training data, a condition that includes a large number of unknown anomaly logs, to measure the generality of the test.

Objective 3: Examination of Log Structure within the dataset.

In this section, we explore the dataset following the grouping of raw logs into fixed groups with a window size of 10. It is essential to examine the types of logs contained in the dataset to evaluate its versatility. For example, an application development site may receive more than 10,000 different types of logs. Therefore, if the experimental dataset comprises only 100 log types, it is inadequate in representing an actual development site, and the accuracy of anomaly detection will be questionable. Thus, this experiment aims to determine the number of log types and their frequency within the dataset.

3. Experimental results

We conducted five trials to eliminate errors in each experiment and evaluated the results using their average.

3.1 Obj1: Generalizability evaluation with cross-validation

Table 2 presents the results obtained through cross-verification applied to the five models. The table reveals the following four observations:

Types of cross verification	F-measure without/with	Recall	Precision
CNN			
Type A	0.217/0.216	0.124/0.122	0.886/0.945
Type B	0.961/0.925	0.992/0.971	0.932/0.885
Type C	0.837/0.698	0.961/0.962	0.750/0.581
Type D	0.718/0.514	0.699/0.151	0.777/0.040
Type E	0.971/0.755	0.996/0.996	0.949/0.636
Ave Without/With	0.741/0.622	0.755/0.683	0.859/0.782
LSTM			
Type A	0.211/0.207	0.118/0.116	0.981/0.975
Type B	0.928/0.908	0.986/0.949	0.883/0.879
Type C	0.540/0.392	0.959/0.963	0.422/0.272
Type D	0.771/0.516	0.784/0.365	0.756/0.877

Types of cross verification	F-measure without/with	Recall	Precision
Type E	0.942/0.909	0.994/0.997	0.895/0.837
Ave Without/With	0.678/0.586	0.768/0.678	0.788/0.768
SPClassifier			
Type A	0.176/0.514	0.279/0.427	0.595/0.696
Type B	0.759/0.689	0.843/0.932	0.695/0.549
Type C	0.568/0.546	0.624/0.777	0.609/0.456
Type D	0.569/0.181	0.426/0.115	0.893/0.675
Type E	0.871/0.840	0.948/0.966	0.809/0.744
Ave Without/With	0.589/0.554	0.624/0.643	0.720/0.634
Auto encoder			
Type A	0.160/0.118	0.954/0.588	0.087/0.080
Type B	0.226/0.214	0.982/0.975	0.128/0.120
Type C	0.019/0.007	0.850/0.669	0.010/0.004
Type D	0.400/0.315	0.666/0.848	0.306/0.218
Type E	0.362/0.152	0.962/0.994	0.232/0.083
Ave Without/With	0.233/0.161	0.883/0.815	0.152/0.101
Transformer			
Type A	0.159/0.159	0.963/0.961	0.087/0.087
Type B	0.457/0.429	0.709/0.712	0.338/0.307
Type C	0.257/0.243	0.521/0.451	0.172/0.166
Type D	0.587/0.508	0.519/0.484	0.675/0.562
Type E	0.865/0.810	0.911/0.911	0.824/0.744
Ave Without/With	0.465/0.430	0.725/0.704	0.419/0.373

Table 2.
Generalizability evaluation results by cross-validation.

1. For all models, the accuracy is notably low when using the Type A dataset.
2. For all models, the accuracy is good when using Type B and Type E datasets.
3. When using the Type D dataset, the accuracy diminishes in supervised learning, whereas unsupervised learning demonstrates higher accuracy compared to the other four types of datasets.
4. When Type A dataset is used, unsupervised learning results in high precision and low recall, while supervised learning results in high precision and low recall.

We attribute observation (4) to the contrasting characteristics of the two learning methods. Supervised learning involves learning from the training data one-to-one, making it proficient in correctly identifying the anomalous data on which it is trained.

Types of cross verification	Normal types Test/validation	Anomaly types Test/validation	Total anomaly logs Test/validation
Type A	681/263	9/8	24,284/ 6881
Type B	64 /27	0 /1	0 /24
Type C	26 / 9	1 /0	24 / 0
Type D	8 /15	0 /0	0 /0
Type E	18 /98	0 /0	0 /0

Table 3.
Composition of unknown logs included in each dataset used for cross-validation.

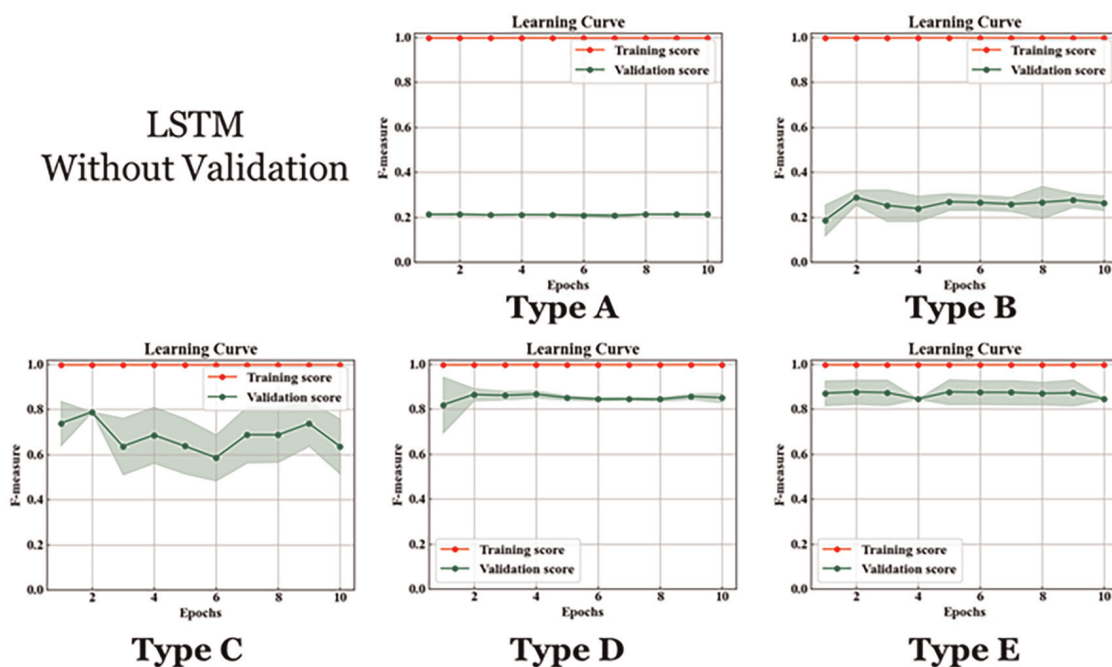


Figure 2.
Learning curve for LSTM.

However, it struggles to detect anomalous patterns outside its learning range, leading to a significant drop in Recall. On the other hand, unsupervised learning often detects patterns that deviate from the training data as abnormal, resulting in higher recall. However, it also tends to identify non-anomalous data as abnormal, leading to a substantially lower precision.

Figures 1 and **2** show the learning curves of the CNN and LSTM models, showcasing their high accuracy. The data points on the graph represent the mean values obtained from five experiments, with the upper and lower widths representing the standard deviations. **Figure 1** demonstrates a substantial variation for Type C, Type D, and Type E.

These findings indicate that the cross-validation results exhibit significant variations in accuracy across different types, suggesting limited generality and versatility.

As an additional investigation, we examined the number of unknown normal and abnormal logs in each split type. The results are presented in **Table 3**. The table reveals that the Type A dataset, which yielded the poorest results, had the highest number and diversity of abnormal logs. Conversely, the datasets used for testing Type

B, Type C, and Type D, which exhibited relatively good results, did not contain any unknown anomaly logs. This suggests that the supervised learning method achieved high accuracy primarily because it was tested with learned anomaly logs. These findings highlight the susceptibility of the system to unknown anomaly logs and indicate that it may not be universally applicable for certain purposes.

3.2 Obj2: Generalizability evaluation with the validation dataset

The comparison between the cases with and without the validation dataset, as shown in **Table 2**, reveals the following characteristics:

1. The CNN, LSTM, and AE models show a decrease in accuracy when the validation dataset is utilized.
2. The Transformer model maintains its accuracy even when the validation dataset is used.
3. The SPClassifier model maintains a basic level of accuracy, but its performance varies depending on the dataset type. Accuracy increases when Type A is used and decreases when Type D is used.

Especially, the accuracy of the CNN and LSTM models decreased by 0.2 when the validation dataset was employed, indicating a potential issue of over-training during the validation phase. Furthermore, the precision values for both CNN and LSTM significantly dropped when validation was used, resulting in a higher number of instances where anomalous data were incorrectly identified as normal. This suggests that over-training may occur during the validation stage, potentially impeding the models' ability to accurately detect anomalies in the test dataset.

The reason behind the observed characteristic (3) can be attributed to the learning method employed by the SPClassifier. In this method, the Spatial Pooling layer generates similar firing patterns for similar inputs and distinct firing patterns for different inputs. Consequently, we posit that utilizing a more complex dataset for validation would lead to a more refined Classifier threshold, ultimately enhancing the accuracy of the model.

3.3 Obj3: examination of log structure within the dataset

The experimental dataset was constructed from the BGL dataset using a sliding grouping method (Window size = 10, Sliding = 1). Sequence Patterns were formed based on templates delimited by Window, and the survey focuses on identifying the number of distinct Sequence Patterns present. **Table 3** showcases a total of 19 Sequence Patterns, which account for 80.3% of the dataset. Considering that there are 131,803 Sequence Patterns in total, the fact that the top 19 patterns represent such a significant portion indicates a bias toward specific Sequence Patterns. It is noteworthy that these 19 patterns are composed of only three templates, suggesting successive repetitions of the same log.

While such Sequence Patterns are commonly observed in OS system logs and network system logs, they are less prevalent in application development, which constitutes a significant portion of software development. In large-scale application development scenarios, where there are tens of thousands of diverse logs, the

simultaneous operation of multiple systems leads to the appearance of complex Sequence Patterns in the output logs. Consequently, a model that exhibits high accuracy on a BGL dataset like this one may not achieve the same level of accuracy when applied to application development due to the inherent differences in the log patterns.

4. Conclusion

In this experiment, our focus was to investigate the limited adoption of deep neural network (DNN)-based anomaly detection methods in the development field. Existing anomaly detection models tend to classify anomaly logs as normal when window grouping is applied. Additionally, when incorporating validation data, the models tend to overfit and exhibit stable learning curves from the initial epoch.

Furthermore, we delved into the structure of the BGL dataset employed in this experiment and observed that certain logs appeared consecutively, with specific Sequence Patterns accounting for a substantial portion of the dataset. In addition, we performed a more in-depth examination of the structure of the BGL dataset used in this experiment. Our findings revealed a recurring occurrence of specific logs within the BGL dataset, along with the presence of certain Sequence Patterns that encompass a significant fraction of the logs. It is crucial to note that in application development, which constitutes a substantial aspect of software development, logs exhibit a greater level of complexity and encompass a wide range of diverse Sequence Patterns. Consequently, the existing representative model faces challenges when applied to the realm of application development.

While the anomaly detection field often focuses on logs with repeated occurrences, such as Super Computer logs or network systems, we aim to target anomaly detection in logs associated with large-scale software development, including applications. Therefore, our plans involve creating diverse datasets that reflect the characteristics of the field under development and exploring the feasibility of employing multiple anomaly detection systems in this context.

Acknowledgements

This work is supported by a grant from Panasonic System Design.

IntechOpen

Author details


Hironori Uchida^{1*}, Keitaro Tominaga², Hideki Itai², Yujie Li¹ and Yoshihisa Nakatoh¹

1 Kyushu Institute of Technology, Fukuoka, Japan

2 Panasonic System Design Co., Ltd., Kanagawa, Japan

*Address all correspondence to: uchida.hironori182@mail.kyutech.jp

IntechOpen

© 2023 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] Chen Z, Liu J, Gu W, Su Y, Lyu M. Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection. United States. Arxiv: <https://arxiv.org/pdf/2107.05908.pdf> [Accessed: 30 April 2023]
- [2] Le V, Zhang H. Log-based anomaly detection with deep learning: How far are we? In: ICSE '22: Proceedings of the 44th International Conference on Software Engineering. Software Engineering. United States: Association for Computing Machinery; 2022. pp. 1356-1367
- [3] He S, Zhu J, He P, Lyu MR. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. United States. 2020. Arxiv Website: <https://arxiv.org/pdf/2008.06448.pdf> [Accessed: 30 April 2023]
- [4] Deep-loglizer. Available from: <https://github.com/logpai/deep-loglizer> [Accessed: 30 April 2023]
- [5] He P, Zhu J, Lyu MR. Drain: An Online Log Parsing Approach with Fixed Depth Tree. 2017 IEEE International Conference on Web Services (ICWS)
- [6] Lu S, Wei X, Li Y, Wang L. Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. 2018 IEEE 16th Intl Conf on D; 2018
- [7] Zhang X, Xu Y, Zhang H, Dang Y, Xie C, Yang X, et al. Robust log-based anomaly detection on unstable log data. In: ESEC/FSE 2019: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Software Engineering. United States: Association for Computing Machinery; 2019. pp. 807-817
- [8] Farzad A, Gulliver TA. Unsupervised log message anomaly detection. ICT Express; 2020;6:229-237
- [9] Nedelkoski S, Bogatinovski J, Acker A, Cardoso J, Kao O. "Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs" 2020 IEEE International Conference on Data Mining (ICDM). Italy: IEEE; 2020
- [10] Hirakawa R, Uchida H, Nakano A, Tominaga K, Nakatoh Y. Large scale log anomaly detection via spatial pooling. Cognitive Robotics. Oct 2021;1:188-196. DOI: 10.1016/j.cogr.2021.10.001
- [11] Cui Y, Ahmad S, Hawkins J. "The HTM spatial pooler—A neocortical algorithm for online sparse distributed coding." Frontiers in Computational Neuroscience. Nov 2017;11. DOI: 10.3389/fncom.2017.00111
- [12] Li L, Zou T, Cai T, Niu T, Zhu Y. A fast spatial Pool learning algorithm of hierarchical temporal memory based on Minicolumn's self-nomination. Computational Intelligence and Neuroscience. 2021;2021. DOI: 10.1155/2021/6680833
- [13] Uchida H, Tominaga K, Itai H, Li Y, Nakatoh Y. Investigation of Weaknesses in Typically Anomaly Detection Methods for Software Development., IHET2023. (in press)