2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC) | 979-8-3503-7696-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/COMPSAC61105.2024.00157

# Unveiling Hidden Anomalies: Leveraging SMAC-LSTM for Enhanced Software Log Analysis

Yicheng Sun, Jacky Keung, Jingyu Zhang*, Hi Kuen Yu, Wenqiang Luo, Shuo Liu
Department of Computer Science, City University of Hong Kong, Hong Kong, China
{yicsun2-c, jzhang2297-c, hikuenyu2-c, wenqialuo4-c, sliu273-c}@my.cityu.edu.hk, jacky.keung@cityu.edu.hk

***Abstract*—Software logs are essential records generated during the functioning of software systems, aiding in the identification of irregularities and prevention of system failures. Recently, deep learning models have garnered significant interest among researchers due to their efficacy in detecting anomalies within software logs. This research paper constructs a novel dataset, consisting of three parts: two datasets derived from our software system, along with a publicly available dataset obtained from the LogHub platform. The extensive logs within the dataset undergo preprocessing to extract meaningful features.**

**Furthermore, this study introduces a novel model named SMAC-LSTM, designed specifically for detecting anomalies in software logs. Sequential Model-based Algorithm Configuration (SMAC) is a suitable method for hyperparameter optimization and automated deep learning. SMAC-LSTM involves determining the optimal hyperparameter values for the LSTM model using the SMAC. Additionally, SMAC-LSTM combines the temporal dependency capturing ability of Long Short-Term Memory (LSTM) with a context-dependent mechanism achieved through a Bayesian optimization algorithm based on random forests. This fusion enhances the model's ability to detect subtle anomalies in time series data, which are frequently disregarded by conventional LSTM models. The thorough evaluation demonstrates the superior performance of SMAC-LSTM models compared to traditional deep learning models, showcasing significant enhancements in precision (98.63%), and recall (92.31%), with an F1-Score of 95.36%, outperforming all other models. These results underscore the potential of SMAC-LSTM in the realm of software log anomaly detection.**

***Index Terms*—Anomaly Detection, Software Engineering, Sequential Model-based Algorithm Configuration (SMAC), LSTM**

## I. Introduction

Software systems are complex and interconnected, relying on each other to process data effectively. In order to ensure high availability and reliability of these interconnected systems, it is crucial to record operational logs [1]. Software logs serve as records generated during the execution of a software system, capturing important information about the system's status, operations, and events [2]. Developers and system administrators play a vital role in monitoring software performance, diagnosing problems, and understanding user behaviors [3]–[5].

To ensure the reliability of systems, logs are widely used as they capture key states during system operation [6], [7]. In recent years, Deep learning-based algorithms have been proposed to address the reliance of machine learning approaches on manually designed feature representations for log data [8]. These algorithms can be categorized into unsupervised [9], [10] , semi-supervised [11], [12], and supervised [13], [14] learning approaches. LogRobust [14] and CNN [15] are notable deep learning algorithms in the supervised learning category, while DeepLog [9], LogAnomaly [16], Logsy [17], and Autoencoder [18] are prominent unsupervised learning algorithms.

In this research, we place our primary focus on evaluating commonly used deep learning models for software log anomaly detection using a meticulously curated dataset. Additionally, we introduce a novel model called SMAC-LSTM, which exhibits significant potential in the domain of software log anomaly detection. Our specific objective is to identify optimal LSTM model parameters and investigate the advantages of the SMAC-LSTM model compared to other deep learning models. In summary, the main contributions of this research are as follows:

- We release a meticulously curated dataset, consisting of a total of 494,191 entries. Within this dataset, 286,371 logs have been sourced from our own software platform, documenting events and triggered exceptions during the software's operation across diverse sectors such as education and e-commerce. We consider this dataset to be extensive in its scope, offering a rich and diverse range of log entries for analysis and evaluation.
- We introduce a novel method, dubbed SMAC-LSTM, demonstrating competitive performance in the realm of software log anomaly detection. By combining LSTM's ability to capture temporal dependencies and context-dependent mechanisms with information integration from a Bayesian optimization algorithm rooted in random forests, SMAC-LSTM enhances sensitivity to subtle anomalies within time series data that conventional LSTM models often overlook.
- To the best of our knowledge, we are the first to employ the SMAC method to optimize and obtain the optimal LSTM parameters for use in log anomaly detection. We conducted a comprehensive comparison of SMAC-LSTM with three foundational deep learning models and carried out an in-depth performance evaluation using binary classification metrics, including ROC, confusion matrices, precision, recall, F1-Score. This analysis highlights the exceptional performance of SMAC-LSTM.

*Corresponding author: Jingyu Zhang.

2836-3795/24/$31.00 ©2024 IEEE
DOI 10.1109/COMPSAC61105.2024.00157

## II. Essentials of Log Processing

Before embarking on the endeavor of log anomaly detection, the creation of a suitable dataset is of paramount importance, as the dataset profoundly impacts data processing and model performance. As shown in Figure 1, the log processing comprises two key steps: log parsing and log representation.

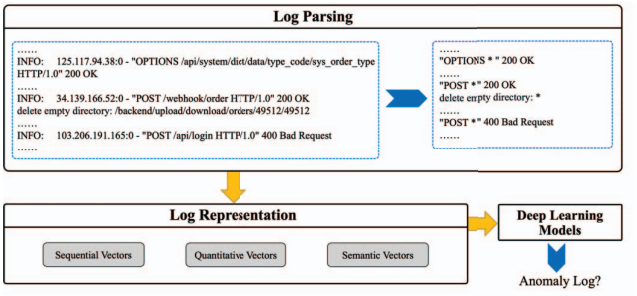


Fig. 1: Related work on log processing.

### A. Log Parsing

Each log entry records a specific system event and may include fields like timestamp, severity level, and message content. To facilitate subsequent modeling, log parsing is a necessary step [19]. Log parsing is employed to automatically convert each log message into a structured format by identifying an event template (constant part) associated with parameters (variable parts) [20]. For example, in a software log message like "*2023-11-17 00:04:09.377 INFO: 34.139.166.52:0 - 'POST /webhook/order HTTP/1.0' 200 OK delete empty directory/backend/upload/download/orders/49512/49512*". After appropriate log parsing, this log message can be split into the event template "'*POST *' 200 OK delete empty directory: **", and the parameter values [*"/webhook/order HTTP/1.0"*, *"/backend/upload/download/orders/49512/49512"*]. Here, "*" denotes the position of a parameter.

### B. Log Representation

Log data must be represented in various formats suitable for deep learning models. Current deep learning-based anomaly detection models typically convert logs into three primary types [1]: (1) Sequential vectors, (2) Quantitative vectors, and (3) Semantic vectors. DeepLog indexes each log event and subsequently produces a sequential vector for individual log windows [9]. LogAnomaly utilizes both sequential and quantitative vectors for anomaly detection [16]. LogRobust employs a pre-trained FastText model to calculate the semantic vectors of log events [14].

## III. Methodology

### A. Long Short-Term Memory (LSTM)

Given that time series data is being used, LSTM networks are suitable as prediction models. LSTM networks employ LSTM units as building blocks in their hidden layers and have demonstrated effectiveness in capturing long-term dependencies [21]. As a variant of Recurrent Neural Networks (RNNs), the distinction between LSTMs and RNNs lies in the structure of each recurrent unit. LSTM consists of three gated structures that control information transmission. These three gates include the input gate $i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$, forget gate $f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$, and output gate $o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$. The input gate regulates the retention of candidate stage data. The forget gate determines the extent to which previous moment internal state information is forgotten. The output gate modulates information flow from the current internal state to the external state.

The weights for input information $x$ are denoted by $W_f$ and $W_o$, while $U_i$, $U_f$ and $U_o$ represent the weights for the previous moment $h_{(t-1)}$. The biases are represented by $b_i$, $b_f$ and $b_o$, and $t$ signifies time.

Wherein, $\sigma$ represents the activation function. In this experiment, the activation function used is the Parametric Rectified Linear Unit (PReLU). PReLU is employed to nonlinearly activate the output features of the convolutional layer and enhances model fitting with minimal additional computational cost and a reduced risk of overfitting.

### B. Bayesian Optimization Algorithm (BOA)

BOA stands as one of the most well-known distribution estimation methods, amalgamating Bayesian networks with evolutionary algorithms to address nearly decomposable problems [22]. BOA extracts comprehensive statistical insights from the best solutions in the ongoing search and represents them using Bayesian networks. As shown in Figure 2, The central concept of BOA involves selecting parameter values that are most likely to facilitate optimization in each iteration, utilizing the current Gaussian process model. This model's prior probability distribution is updated through the application of the Bayesian theorem, leading to the creation of the posterior probability distribution through stochastic sampling and function evaluation [23]. In comparison to conventional search algorithms, BOA offers swifter search speeds and requires fewer iterations, making it advantageous for optimizing hyperparameters in machine learning algorithms [24].

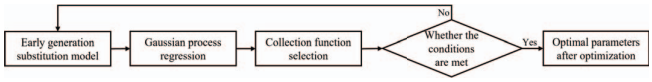


Fig. 2: The process of Bayesian Optimization Algorithm.

Neural networks encompass various hyperparameters that, in traditional LSTM models, are frequently determined based on empirical knowledge. Identifying the most appropriate parameters for the model through empirical methods can be challenging. These hyperparameters significantly impact the runtime and prediction accuracy of neural networks; thus, optimizing them is crucial.

### C. Sequential Model-based Algorithm Configuration (SMAC)

Conventional Gaussian process-based Bayesian optimization is limited to handling continuous or numerical variables in the parameter space [25]. However, in the realm of deep

learning, there are numerous parameters that are discrete or exhibit conditional relationships with each other. As an alternative to Gaussian Processes, Random Forest regression has been proposed as a versatile surrogate model within the framework of SMAC [26]. SMAC is a suitable method for hyperparameter optimization and automated deep learning. It employs a model-based search strategy using Random Forests as its surrogate model to approximate the optimization of the objective function. While Bayesian optimization is typically associated with Gaussian Processes as the surrogate model, SMAC has demonstrated effectiveness in handling problems involving discrete and conditional hyperparameter spaces using Random Forests.

SMAC constructs its surrogate model using Random Forest, which excels in dealing with discrete inputs. The Random Forest comprises multiple decision trees that contribute to the overall prediction by computing mean and variance from the output of each tree. These ensemble predictions serve as the output of the surrogate model, offering estimates of performance metrics for various hyperparameter configurations.

### D. The Proposed Method: SMAC-LSTM

In this paper, we introduce the SMAC-LSTM method. As shown in Figure 3, the application of this software anomaly decision method comprises three primary steps.

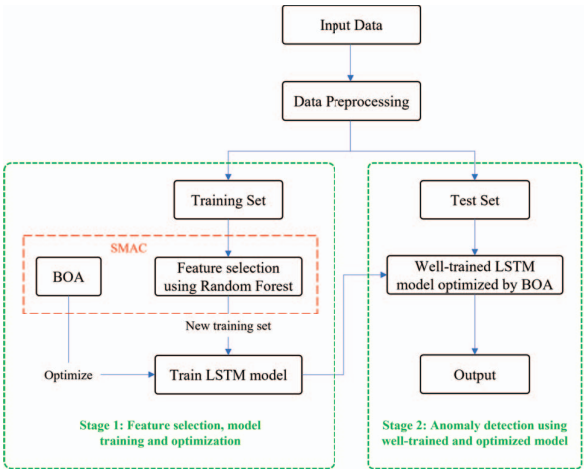


Fig. 3: Flowchart of the SMAC-LSTM Method.

**Step 1**: Employ the random forest algorithm to select feature variables from software logs.

*(1.1)* Construct a training set using initial variables.

*(1.2)* Utilize random forest to assess the significance of initial variables and document the classification accuracy of the training set data. This step focuses on employing the random forest model to assess the contribution of each variable to the model's predictive capabilities. The algorithm produces a list of variable importances, with each variable's importance based on its average reduction in impurity across multiple decision trees. Essentially, the goal is to understand the impact of variable significance on the model's predictions.

*(1.3)* The model attains its highest classification accuracy when the current set of input variables is deemed significant for the predictive task.

**Step 2**: Train the LSTM network using the training set with feature variables and optimize its parameters using BOA.

**Step 3**: Feed the feature variable data from the test set into the well-trained LSTM network and determine the presence of anomaly logs based on the model's output.

Overall, LSTM functions are the foundational model in this methodology, utilized to detect software logs through learning from feature variable data. BOA is applied to fine-tune the hyperparameters of LSTM, thereby improving its efficacy. The input for LSTM is determined using feature parameters selected by random forest, thereby reducing the influence of superfluous parameters and enhancing the classification efficiency of LSTM. For successful feature selection, establishing a comprehensive initial set of parameters is crucial.

## IV. Study Design

### A. Dataset Construction

To assess the performance of the deep learning-based anomaly detection model, we assembled three datasets, as outlined in Table I, namely OpenStack, Android, and Apache, containing 494,191 data entries in total. The OpenStack dataset originates from an open dataset on LogHub [27], while the other two datasets were extracted from the log files of software platforms supported by our industrial partners.

TABLE I: Summary of datasets used in the study.

| Dataset | Category | Numbers | Anomalies | Source |
|---|---|---|---|---|
| ***OpenStack*** | Distributed system | 207,820 | 8.87% | LogHub |
| ***Android*** | Mobile system | 83,248 | 2.99% | Industry |
| ***Apache*** | Web server application | 203,123 | 4.43% | Industry |

### B. Data Preprocessing

Before initiating our experiments, it is necessary to preprocess the log data within the dataset, adhering to the process mentioned previously. Our fundamental operations are as follows:

**Log parsing**. We first extract log templates from the log data, using the log parser Drain [28] with default parameter settings. The log data is denoted by $L = \{l_1, l_2, ..., l_i, ..., l_{N_L}\}$ and contains $N_L$ entries (lines) of log messages. Each log message $l_i$ is parsed into a log template $E(L_i)$, which is denoted $e_i$ for short.

**Log representation**. We apply an embedding layer for transforming categorical log entries into dense, high-dimensional semantic vectors. This technique, inspired by word embeddings in NLP, allows the capture of intricate contextual relationships within software logs. Each unique identifier in the logs, such as error codes or event types, is associated with a vector that is fine-tuned during the neural network's training process. These semantic vectors enrich the log data representation, facilitating more sophisticated pattern recognition and improving the model's ability like anomaly detection and classification with higher accuracy.

### C. Model Parameters

As depicted in Figure 4, a simple LSTM network architecture is employed, comprising input layer, LSTM layer, fully connected layer, softmax layer, and output layer.


Fig. 4: A simple LSTM network architecture.

The input layer consists of 32 nodes, representing the input for the anomaly log text, which comprises 32 characters. The LSTM layer, also known as the hidden layer, contains multiple LSTM units. Following this is a fully connected layer with 2 output nodes, corresponding to the two classes for prediction. To predict class labels, the network concludes with a softmax layer and classification output layer. The output of the fully connected layer is classified by the softmax function into either 0 or 1. In the softmax layer, classification probabilities are calculated, and the classification output layer categorizes the test data into two groups: events and non-events. The challenge of anomaly detection in software logs is typically framed as a binary classification problem. Therefore, A binary cross-entropy loss function is chosen as the cost function.

The weights and biases of the LSTM network are initialized randomly, and the Adaptive Moment Estimation (Adam) algorithm is employed to optimize the internal parameters of the LSTM network during the training process.

Similar to the LSTM, the SMAC-LSTM model incorporates certain parameters present in the LSTM model. The difference lies in the use of BOA within the SMAC method to determine hyperparameters such as *batch_size*, *learning_rate*, *epochs*, *embedding_dim*, *hidden_size*, *num_layers*, and *dropout*. As shown in Table II, establishing specific parameter ranges for optimization is crucial to achieve effective optimization results and ensure the feasibility of the optimized parameters.

TABLE II: Parameters range of BOA.

| Parameters | Value/Range |
|---|---|
| batch_size | 32, 64, 128, 256 |
| learning_rate | [0.0001, 0.01] |
| epochs | 20, 40, 60, 80, 100 |
| embedding_dim | 32, 64, 128 |
| hidden_size | 32, 64, 128 |
| num_layers | 1, 2 |
| dropout | [0, 0.5] |

### D. Comparative Experiment

To validate the effectiveness of our approach, we conducted experiments comparing the SMAC-LSTM model to three common deep learning models (RNN, GRU, LSTM). These models were tested on the dataset created, with an 80% portion for training and a 20% portion for testing. The goal was to determine whether our SMAC-LSTM model could be considered the optimal choice for deployment in our production environment. For the LSTM model, setting its parameters posed a challenge because, within our SMAC-LSTM model, the parameters of the LSTM are determined iteratively by BOA. This aspect will be further detailed in the analysis section.

To ensure robustness and mitigate randomness, we conducted each experiment five times and reported the average results. All experiments were performed on a system running Windows 11 with an Intel(R) Core(TM) i7-12700 CPU, 64GB RAM, and an NVIDIA RTX A4000 GPU.

### E. Evaluation Metrics

Anomaly detection is treated as a classification problem. To assess the effectiveness of models in detecting anomalies, we rely on $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, and $F1\text{-}score = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$ metrics to evaluate model performance.

TP (True Positive) is the count of anomalous log sequences correctly detected by the model. FP (False Positive) is the count of normal log sequences incorrectly identified as anomalies. TN (True Negative) is the count of normal logs that are correctly classified. FN (False Negative) is the count of anomalous log sequences that the model failed to detect. The above four states (TP, FP, TN and FN) can be more effectively visualized and represented using a confusion matrix.

## V. Results and Analysis

### A. Determination of Model Parameters

The initial step in the SMAC-LSTM model involves determining the optimal hyperparameter values for the LSTM model using the SMAC method. A total of 20 iterations were set for this process. The combination of hyperparameters is initially chosen by BOA, and the random forest is used to assess the significance of initial variables. The classification accuracy of the training dataset is recorded, and the optimal set of hyperparameters is determined through iterative steps. Ultimately, at the 16th iteration, an ideal combination of hyperparameters is achieved, as displayed in Table III. It is worth noting that BOA operates swiftly as it establishes an alternative hyperparameter search space model and conducts its search within this dimension, rather than the original actual hyperparameter space. In conclusion, the most ideal combination of hyperparameters is preserved, representing the LSTM model parameters derived through the SMAC method.

TABLE III: LSTM Parameters Optimized by SMAC.

| Parameters | Value |
|---|---|
| batch_size | 256 |
| learning_rate | 0.00192 |
| epochs | 80 |
| embedding_dim | 64 |
| hidden_size | 64 |
| num_layers | 1 |
| dropout | 0.22962 |

### B. Comparative Experiment Analysis

In this section, we evaluate the effectiveness of three prevalent deep learning models (RNN, GRU, LSTM) on our dataset and compare them with our SMAC-LSTM model to determine the most suitable candidate for deployment in our production environment.

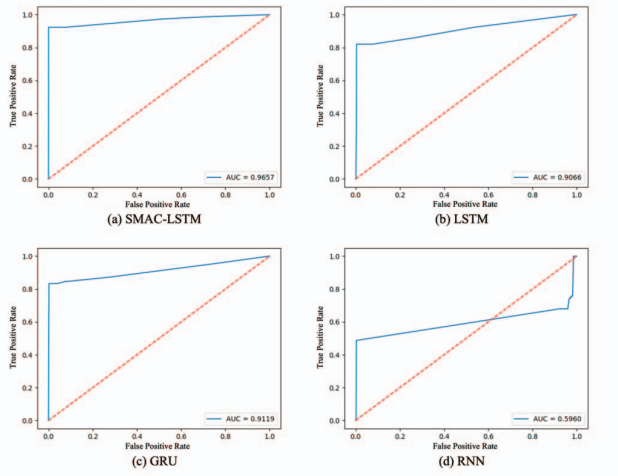


Fig. 5: ROC curves of each model.

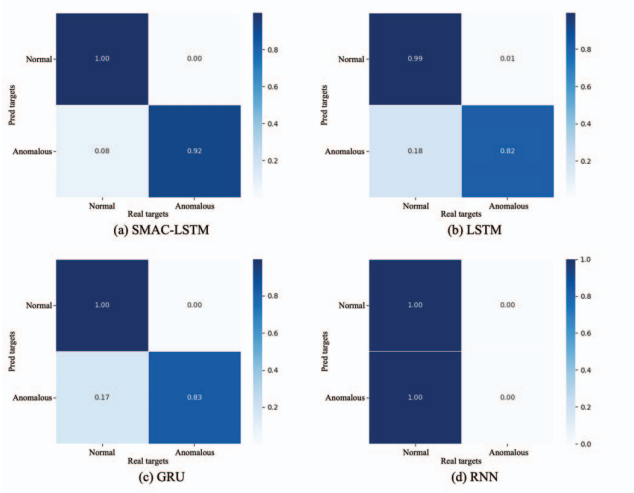


Fig. 6: The confusion matrix of each model.

Figure 5 displays the ROC curves and AUCs for these four different models. It is essential to note that since the parameters of LSTM within SMAC-LSTM are optimized by BOA, setting these parameters becomes a challenge in designing comparative experiments. To better demonstrate the effectiveness of the SMAC method for LSTM results, we utilize each set of optimized hyperparameters found by SMAC for LSTM (totaling 20 sets) as input and ultimately select the best-performing set for comparison. As observed in the graph, the ROC curve of SMAC-LSTM is positioned closer to the top-left corner compared to the other models, with an AUC of 0.9657, surpassing the rest. This indicates that SMAC-LSTM can achieve a better balance between Detection Rate (DR) and False Alarm Rate (FAR). In other words, for the same FAR, it achieves a higher DR, and for the same DR, it maintains a lower FAR. Therefore, when compared to the other models, SMAC-LSTM performs better in terms of AUC. Specifically, the AUCs of LSTM, GRU, and RNN are 0.9066, 0.9119, and 0.596, respectively. This suggests that both LSTM and GRU can predict anomaly logs effectively, but RNN's performance is not as impressive. The primary reason for this is that software anomaly logs are often scarce, causing RNN to struggle in accurately identifying these anomalies, as explained further in the analysis section. In summary, the feature variables selected by the random forest within the SMAC method can enhance the classification performance of LSTM to some extent, and the optimal hyperparameters derived from BOA significantly improve LSTM's performance.

The confusion matrices for the four models are depicted in Figure 6. To enhance clarity, we have normalized the confusion matrix so that the sum of each row equals 1. From the figure, it is evident that the sum of TP and TN is highest for SMAC-LSTM, and SMAC-LSTM also having the highest TP, indicating its ability to identify the most anomaly logs.

Given the scarcity of anomaly logs in the dataset, even a single misjudgment can significantly impact the FN. FN signifies the model incorrectly identifying an anomaly log as normal, and SMAC-LSTM's performance is exceptional, notably surpassing other models. RNN, with FN equal to 1, suggests that due to the limited anomaly logs, RNN failed to recognize any anomaly logs and misclassified all of them as normal. This situation highlights RNN's unsuitability for detecting anomaly logs. Additionally, SMAC-LSTM has a FP count of 0, whereas LSTM does not, indicating that the SMAC method has enhanced LSTM's performance by preventing it from misclassifying normal logs as anomalies. Overall, SMAC-LSTM maintains the highest performance.

TABLE IV: Evaluation criteria values of each model.

| Models | AUC | Precision | Recall | F1-Score |
|---|---|---|---|---|
| SMAC-LSTM | 0.9657 | 98.63% | 92.31% | 95.35% |
| LSTM | 0.9066 | 70.33% | 82.05% | 75.74% |
| GRU | 0.9119 | 82.28% | 83.33% | 82.80% |
| RNN | 0.5960 | 0 | 0 | 0 |

As shown in Table IV, SMAC-LSTM achieves a very high precision rate (98.63%) and a substantial recall rate (92.31%), with an F1-Score of 95.36%. Despite the limited quantity of anomaly logs, SMAC-LSTM performs well, demonstrating its robustness in handling imbalanced datasets and its reliability in detecting anomaly logs. It is clear that SMAC-LSTM outperforms all other models. The standalone LSTM model's performance is not particularly ideal, as its F1-Score is 75.74% lower than that of GRU. However, with the incorporation of the SMAC method, SMAC-LSTM exhibits significant improvement, outperforming GRU and increasing its F1-Score by 19.61% compared to standalone LSTM. This demonstrates the substantial enhancement of LSTM's performance through the feature variables captured by the random forest and the optimal hyperparameters derived by BOA in the SMAC method.

In contrast, RNN performs poorly. In a well-functioning software system, errors are infrequent or even non-existent, resulting in a low number of anomaly logs. Consequently,

RNN struggles to recognize the characteristics of anomaly logs, resulting in TP equal to 0. This further emphasizes the challenges associated with software anomaly log detection.

In summary, SMAC-LSTM demonstrates exceptional performance across both categories, even considering the limited sample size of the anomaly logs. These results are impressive, indicating the model's strong generalization ability and sensitivity to different categories, making it a valuable asset for practical applications. When compared to other models, our SMAC-LSTM model delivers outstanding performance and is the best candidate for our dataset.

## VI. Conclusion

The SMAC-LSTM introduced in this paper has showcased its tremendous potential in the domain of software log anomaly detection. By leveraging the strengths of LSTM in capturing temporal dependencies and incorporating context-dependent mechanisms through a Bayesian optimization algorithm based on random forests, SMAC-LSTM has demonstrated enhanced sensitivity to subtle anomalies within time series data that are often overlooked by traditional LSTM models.

## Acknowledgement

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong and the research funds of the City University of Hong Kong (6000796, 9229109, 9229098, 9220103, 9229029).

## References

[1] 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.

[2] J. Nyyssölä and M. Mäntylä, "Efficiency of unsupervised anomaly detection methods on software logs," *arXiv preprint arXiv:2312.01934*, 2023.

[3] B. Yu, J. Yao, Q. Fu, Z. Zhong, H. Xie, Y. Wu, Y. Ma, and P. He, "Deep learning or classical machine learning? an empirical study on log-based anomaly detection," in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2023, pp. 392–404.

[4] A. R. Chen, "An empirical study on leveraging logs for debugging production failures," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 126–128.

[5] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Machine Learning with Applications*, vol. 12, p. 100470, 2023.

[6] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[7] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 2007, pp. 583–588.

[8] C. D. Manning, *An introduction to information retrieval*. Cambridge university press, 2009.

[9] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[10] V. Zeufack, D. Kim, D. Seo, and A. Lee, "An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis," *High-Confidence Computing*, vol. 1, no. 2, p. 100030, 2021.

[11] Y. Lin and Y.-Y. Chiang, "A semi-supervised learning approach for abnormal event prediction on large network operation time-series data," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 1024–1033.

[12] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.

[13] S. Ying, B. Wang, L. Wang, Q. Li, Y. Zhao, J. Shang, H. Huang, G. Cheng, Z. Yang, and J. Geng, "An improved knn-based efficient log anomaly detection method with automatically labeled samples," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 3, pp. 1–22, 2021.

[14] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *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*, 2019, pp. 807–817.

[15] 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)*. IEEE, 2018, pp. 151–158.

[16] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

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

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

[19] X. Wang, X. Zhang, L. Li, S. He, H. Zhang, Y. Liu, L. Zheng, Y. Kang, Q. Lin, Y. Dang *et al.*, "Spine: a scalable log parser with feedback guidance," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1198–1208.

[20] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[21] L. S.-T. Memory, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 2010.

[22] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[23] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian hyperparameter optimization on large datasets," 2017.

[24] Q. Shang, D. Tan, S. Gao, L. Feng *et al.*, "A hybrid method for traffic incident duration prediction using boa-optimized random forest combined with neighborhood components analysis," *Journal of Advanced Transportation*, vol. 2019, 2019.

[25] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*. Springer, 2011, pp. 507–523.

[26] ——, "Sequential model-based optimization for general algorithm configuration (extended version)," *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.*, 2010.

[27] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-driven log analytics," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 355–366.

[28] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.