Assessing Anonymized System Logs Usefulness for Behavioral Analysis in RNN Models

Tom Richard Vargis[1,*], Siavash Ghiasvand[1,2]

[1]Technische Universität Dresden, Germany
[2]Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI) Dresden/Leipzig, Germany

Abstract

System logs are a common source of monitoring data for analyzing computing systems behavior. Due to the complexity of modern computing systems and the large size of collected monitoring data, automated analysis mechanisms are required. Numerous machine learning and deep learning methods are proposed to address this challenge. However, due to the existence of sensitive data in system logs their analysis and storage raise serious privacy concerns. Anonymization methods could be used to cleanse the monitoring data before analysis. However, anonymized system logs in general do not provide an adequate usefulness for majority of behavioral analysis. Content-aware anonymization mechanisms such as *PαRS* preserve the correlation of system logs even after anonymization. This work evaluates the usefulness of anonymized system logs of Taurus HPC cluster anonymized using *PαRS*, for behavioural analysis via recurrent neural network models. To facilitate the reproducibility and further development of this work, the implemented prototype and monitoring data are publicly available [12].

Keywords

System log analysis, Data usefulness, Time series analysis

1. Introduction

It is of great interest to monitor large computing systems' behavior. This knowledge helps to improve availability, reduce further damages caused by detectable failures, and diagnose problems. Despite the independent functionality of computing nodes in large computing systems, their behavior is highly dependent on other components of the computing system. The hierarchical design (e.g., Fat tree topology) of large computing systems, such as high-performance clusters (HPC), and the utilization of shared resources in such systems are the main reason for behavioral dependency among the computing nodes. Furthermore, the strategies employed to determine the usage of node sets to further enhance system performance (e.g., utilizing neighboring nodes) also have a direct impact on behavioral dependencies among computing nodes. Earlier studies identified strong spatial and temporal correlations among computing nodes of large computing systems [1].

In recent years, numerous automatic and semi-automatic behavioral analysis methods have been proposed. These methods utilize various monitoring data such as data collected by hardware sensors, system logs, batch system information, and user activity logs to detect and

International Workshop on Data-driven Resilience Research 2022, July 6, 2022, Leipzig, Germany
*Corresponding author.
tom_richard.vargis@tu-dresden.de (T. R. Vargis); siavash.ghiasvand@tu-dresden.de (S. Ghiasvand)

predict the system behavior. Due to the complexity of large-scale computing systems and their dynamic nature, identifying a system's behavioral pattern is challenging. Furthermore, sensitive information contained in the monitoring data (e.g., user activity logs), has raised privacy concerns about the use of some analytical methods as well as the outsourcing of analysis.

The anonymization method PαRS [2] has been proposed to address the privacy concerns of processing monitoring data containing sensitive information. Preliminary results indicate the usefulness of such anonymized system logs for the detection of abnormal behaviors in HPC systems via auto-encoders [3]. This study examines the effectiveness of using fully anonymized system logs in one of the most commonly used models of recurrent neural networks (RNN) for anomaly detection, namely Long short-term memory or LSTM. Due to the nature of these analyses, a short return time, as well as a short training time, is required. Therefore, the model needs to be kept as simple as possible, and it should be possible to train the model with as little data as possible.

The remainder of this work is structured as follows: Section 2 provides an overview of using deep learning methods for behavioral analysis, with a focus on LSTM. Section 3 describes the monitoring data, the preprocessing steps, and the main parameters used in this work. The fitness of the proposed model for this work is verified in Section 4. In Section 5 the results of each experiment have been discussed. Finally, Section 6 concludes the work and specifies the important future work directions.

2. Related works

Anomaly detection is a pivotal part of system log analysis and has been the subject of numerous types of research. Among common deep learning models for anomaly detection, LSTM has been widely employed due to its success in providing highly accurate predictions. Qicheng Ma et al. in [4] and Min Du et al. in [5] modeled system logs as natural language sequences and patterns were extracted from these sequences. This analysis was done to detect insider threats and any deviations from these sequences were seen as a potential threat. A similar approach was used in [6] which had a feature extraction algorithm like Word2vec and then employed an LSTM for anomaly detection.

In [7] log patterns from heterogenous logs were extracted by clustering similar logs together and from these patterns, sequential features over time were extracted. These features over time were finally passed through LSTM to detect failures. Log parsing and feature extraction followed by two LSTMs and an Autoencoder was introduced in [8] for failure detection. An abnormal instance usually manifests itself as an outlier that significantly deviates from such patterns. Zhuangbin et. al. concluded that log semantics indeed improves models' robustness against noises.

Hao Chen et al. in [9] proposed a novel semantic information embedding technique to detect anomalies. Some keywords in log entries may represent the meaning of the entire system logs. A CNN combined with the LSTM approach not only learns semantics but also the quantitative feature from the log count vector. A slightly different approach is proposed in [10]. Yixin et. al. in addition to the deep learning model takes a step further by closely examining the

timestamps of the log data which majority of existing studies have generally ignored. Yixin et. al. propose to integrate log timestamps in deep learning models using interpolation techniques. This addition was proved to improve the ultimate accuracy of failure detection.

The above-mentioned studies deliver a detection accuracy of greater than 93%[1]. Whilst some of these approaches use a supervised method where predefined ground truth is set as the pattern and any deviation from these were classified as an anomaly, some others use an unsupervised approach where patterns were identified from the extracted sequential features by the monitoring data over time. The common point in all the above studies is the usage of monitoring data in its original format. The existence of sensitive information in monitoring data raises serious concerns in many use cases, the storage of monitoring data becomes challenging and the outsourcing of analysis is not possible.

Conversely, to the aforementioned studies, the approach proposed in this work employs fully anonymized system logs. Thus, eliminating all privacy concerns and making it possible to outsource the entire log analysis process. On the other hand, the usage of anonymized monitoring data makes the identification process increasingly challenging since the content of the encoded logs cannot be retrieved.

3. Method and Data

Recurrent Neural Networks (RNN) are known to perform well on time series data. The model, Long Short-Term Memory (LSTM) is a special type of RNN capable of learning dependencies between the data and making sequential predictions. This work builds an LSTM model that requires short memory of the past to identify and predict the pattern of upcoming log messages. The log messages are anonymized in a pre-processing step based on the PαRS mechanism [2]. This is achieved by classifying messages with similar patterns into a single class (pattern) and then generating a unique hash key for each pattern. Both univariate and multivariate data were tested on this model. The final goal is to assess the usefulness of anonymized system logs for anomaly detection via LSTM models. Thus, it is essential to avoid unnecessary complexities. To achieve this goal, a one-layer LSTM model followed by one dense layer was selected. The Keras[2] library was used to implement the model.

In this work, Adam is used as the optimizer and the Mean Absolute Error (MAE)is employed as the loss function. Mean Squared Logarithmic Error (MSLE) is also used in the later parts of the report. MSLE considers the relative difference between the real and the predicted value. As the data used for the analysis is normalized, choosing MAE over MSLE is not expected to make a notable difference to the error values. In the testing dataset, a point is classified as an anomaly if the MAE loss goes beyond the specified threshold. Here the threshold is defined as the maximum value of the MAE loss for the training dataset.

System Logging Protocol or Syslog is the common protocol used to send system logs or event messages to a specific server, called a Syslog server. It is primarily used to collect various device logs from several machines in a central location for monitoring and review. Syslog is available

[1]In controlled environment and with adequate data preparation steps, near to perfect accuracy is possible.
[2]Available at https://keras.io/about/.

in all Unix and Linux-based systems. As all the TOP500[3] HPC systems are Linux-based (at the time of writing), Syslog analysis applies to all HPC systems. The detailed specification of the Syslog protocol is defined in RFC5424[4].

Taurus[5] HPC cluster, located in Dresden, has around 2000 compute nodes including 750 GPUs and a total count of 80,000 CPU cores. Taurus is divided into several sections known as Islands. Each Island has its specific hardware configuration. Island 8 of Taurus is powered by AMD Rome CPUs and NVIDIA A100 GPUs. Thus, one of the most utilized islands on Taurus by numerous users and for various applications. For this work, the system logs of first the 16 nodes of Island 8 are used as the source of monitoring data. This selection is based on the idea that Island 8 of Taurus is an active Island and the first 16 nodes are known to have shared resources.

For the multivariate model, four features were considered in a specified time bucket (eg: 10 min), namely the average severity, average facility, the frequency of top 10 messages[6] from the last 24 hours, and the frequency of non-top 10 messages from the last 24 hours.

Furthermore, for the univariate model two synthesized datasets, one with repetitive patterns and the other with significant noises were examined. The goal of using synthesized datasets was to identify the similarities in the model's behavior based on the input data.

The model and the collected monitoring data have several adjustable parameters which can be fine-tuned to improve the prediction accuracy. Table 1 provides a list of parameters used in this work.

Parameter	**Possible values**
Stateful	Fixed
Node bucket	Fixed (16 nodes Island8)
Learning rate	0.001, 0.01, 0.1
Epochs	20, 50, 100, 10000
Number of steps	2, 4, 6, 12
Time bucket	5 min, 10 min, 30 min
Cumulative Sum	Yes, No
Normalization	None, MinMax, Sigmoid
Number of top messages considered	Top3, Top5, Top10

Table 1: List of metrics and hyper-parameters

The *learning rate* hyperparameter used in the training of the LSTM is given a value between 0.0 to 1.0. Four different learning rates [0.1,0.01,0.001,0.0001] were used in the first experiments. However, to better observe the relevance and impact of learning rate[7], the three larger rates were used in most experiments. Although in the Keras[8] framework, by default, LSTMs are stateless, to consider dependencies between batches, the model needed to be stateful. The *time*

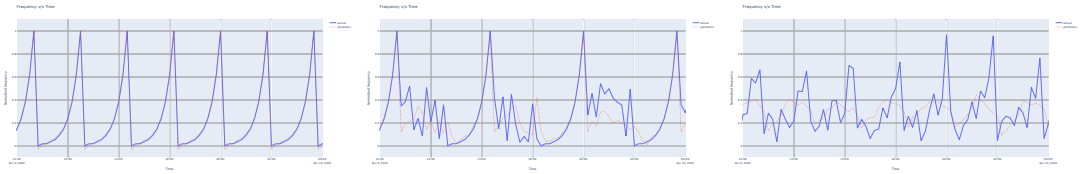
[3]https://top500.org
[4]https://datatracker.ietf.org/doc/html/rfc5424
[5]Detailed hardware information: https://tud.link/7y2h
[6]The top 10 classes of syslog messages with highest frequency.
[7]From various studies it is known that large values of learning rate may impose destructive impact on the learning process.
[8]Available from https://keras.io/.

bucket shows the time interval in which the monitoring data is aggregated. For practical reasons detecting anomalies with a delay of more than 30 minutes is not favorable. The *number of steps* in principle is the memory of the LSTM. The model uses this number of steps to predict the following step. For example, if the *time bucket* is given as 10 minutes and the number of steps as 6, then 60 minutes of data is used to predict the next data point. *Node bucket* defines the number of nodes selected from an Island on Taurus. Considering the significant similarity in behavior of the neighboring nodes[9] due to shared resources, the first 16 nodes of Island 8 are taken for training the model.

Cumulative sum of the count of messages is an additional setup where the frequency features are cumulatively summed up every hour. This accumulation amplifies the system logs' hourly pattern facilitating identification of the same by the model. Data is *normalized* either using the MinMax scaler function which normalizes the dataframe according to the maximum value or via a sigmoid function which adjusts the data to the scale of 0 to 1.

4. Model fitness

To evaluate the fitness of the proposed model for the intended purpose of this work, a synthesized dataset with repetitive patterns was constructed. The synthesized dataset consisted of the first 10 Fibonacci numbers repeated 100 times. The proposed model was able to identify and learn the repetitive pattern and predict the series with high accuracy as shown in Figure 1a.



(a) Fibonacci dataset (b) Partially randomized (c) Detectable trends

Figure 1: Assessment of LSTM model using synthesized dataset

Furthermore, this data was slightly altered by having the first 10 Fibonacci numbers followed by 10 random numbers from the range of 0 to 30. This set of 20 numbers was then repeated 100 times, to form the test dataset. Figure 1b shows that the model successfully recognized the repetitive pattern of the Fibonacci part and resembled the random part of the data with a short delay.

Finally, a random noise (in the range of 0 to 30) was added to the initial dataset and the same model was trained using this new dataset. This time, as shown in Figure 1c the predictions lag and, as expected, the model could only learn the general trend in data.

From these observations, it is concluded that the proposed model can learn patterns and trends in data despite the small size of the dataset, and the low number of epochs. It is important to note that most applications of log analysis (e.g. live anomaly detection) require a short execution time. Therefore, using small datasets and a low number of epochs is highly favorable.

[9] also known as node vicinity [11]

Univariate

Time bucket			10-min				30-min			
Number of steps			2	6	9	12	2	6	9	12
Batch size			134	78	89	622	125	27	206	123
		Lr								
Loss value	Training	0.001	0.050	0.041	0.047	0.061	0.074	0.070	0.068	0.070
	Validation		0.057	0.049	0.052	0.081	0.088	0.076	0.091	0.087
	Training	0.01	0.051	0.038	0.033	0.044	0.069	0.055	0.058	0.058
	Validation		0.057	0.042	0.037	0.058	0.083	0.065	0.079	0.066
	Training	0.1	0.066	0.218	0.603	0.082	0.068	0.506	1.846	0.082
	Validation		0.085	0.314	0.693	0.100	0.081	0.796	2.159	0.099

Multivariate

Epochs			50		100	
Number of steps			2	6	2	6
Batch size			199	1	125	27
		Lr				
Loss value	Train.	0.0001	0.147	0.124	0.145	0.121
	Val.		0.114	0.077	0.113	0.079
	Train.	0.001	0.149	0.113	0.139	0.092
	Val.		0.112	0.074	0.109	0.080
	Train.	0.01	0.140	0.114	0.145	0.115
	Val.		0.105	0.085	0.108	0.078

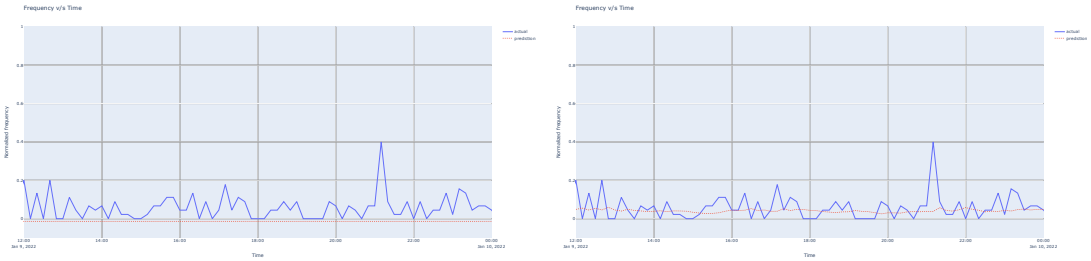
Table 2: Comparison of loss values for varying learning rates and number of steps

5. Results and Discussions

Different combinations of parameters were tested for both univariate and multivariate datasets to evaluate their impact on the model's prediction. The results of both uni- and multivariate datasets are shown in Table 2.

For the univariate dataset, in each experiment, one parameter takes different values from the possible values, while the other parameters remain constant. The set of parameters used in each experiment is shown as a tuple. The "*" represents the varying parameter.

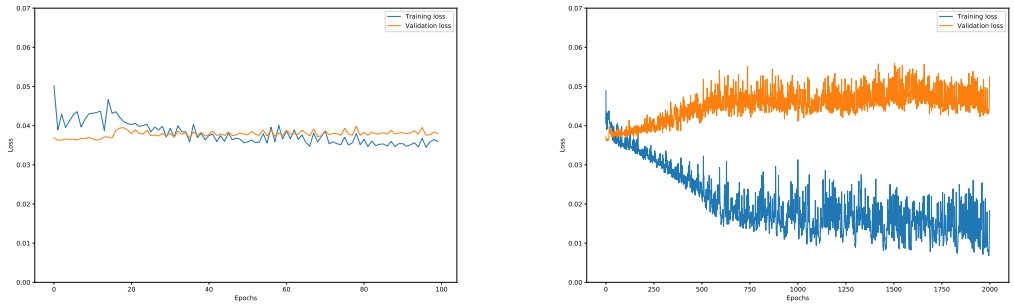
Learning rate (Lr) $[*, 100, 6, 10min, No, MinMax, top10]$
While at a learning rate of 0.1 the model fails as expected (Figure 2a), it yields slightly better predictions for a learning rate of 0.001 (Figure 2b).



(a) Lr = 0.1 (b) Lr = 0.001

Figure 2: Impact of learning rate on predictions (time bucket = 10 min)

Epochs $[0.01, *, 6, 10min, No, MinMax, top10]$
With a low number of epochs, e.g., 20 epochs, sometimes the error is not sufficiently small and with large epochs, e.g., 100 or 200 epochs, the error starts to oscillate towards the end. There is no significant change observed in model behavior for epochs beyond 100. A comparison between 100 and 10000 epochs is shown in Figure 3. The overfitting problem with a large number of epochs is visible in Figure 3b. Similar observations were made in all experiments. Therefore, the decision was made to fix the epochs at 50 so that the error converges.



(a) Epoch = 100 (b) Epoch = 10,000

Figure 3: Impact of different amount of epochs on loss value

Number of steps (Ns) $[0.01, 50, *, 10min, No, MinMax, top10]$
Although the number of steps is an adjustable parameter it is also connected to the time bucket size. Six steps for a 10 min time bucket means that the model uses 60 minutes of data (6 rows of data) to predict the next row. Training on a few steps, e.g., 2 steps, reduces the model's memory. Thus, instead of predicting future behavior, the model memorizes the patterns observed in recent time steps. In contrast, using a large number of time steps, e.g., 12 steps, the model will be able to generalize and predict the overall trend in data.

Time bucket $[0.01, 50, 6, *, No, MinMax, top10]$
The data collected within a time bucket of 5 minutes contains too many null values leading to wrong predictions. The 10 and 30-minute buckets provide relatively better data (less null values). In addition, Taurus' behavior is highly dependent on the behavior of its users. Hence, it shows hourly and daily patterns.

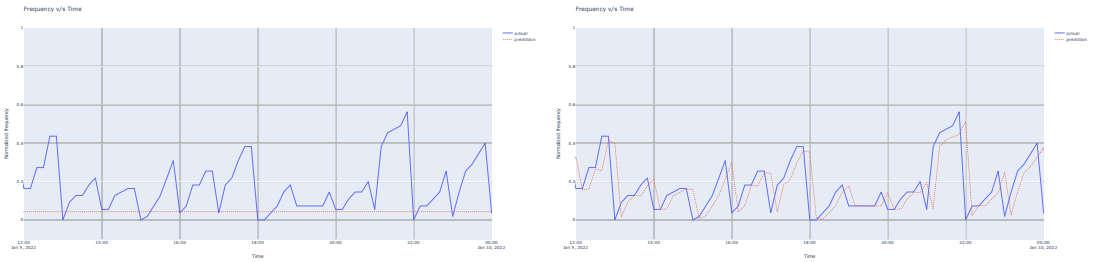
Cumulative Sum $[0.01, 50, 6, 30min, *, MinMax, top10]$
There is a significant improvement in predictions when the cumulative sum is introduced in any setup. This is possibly due to the hourly patterns (influenced by user activities) which are known to exist within Taurus log messages.

Normalization $[0.01, 50, 6, 30min, Yes, *, top10]$
Using non-normalized data versus MinMax scaled data, slightly changes the final output. Sigmoid normalization does not improve the predictions hence, these are excluded. Since an effective improvement cannot be detected, the data in this work is always normalized with the MinMax scaler.

Number of top messages considered $[0.01, 50, 6, 30min, Yes, MinMax, *]$
The choice of the number of top messages slightly influences the predictions. Considering fewer top messages makes the predictions marginally better. However, this improvement is not significant enough to vary this parameter.

Univariate data From the above observations the parameter list to be varied is narrowed down to the Learning rate, Time bucket, and Number of steps. Both time buckets of 10 minutes and 30 minutes are considered. The learning rate and the number of steps are varied among the selected range. The batch size is automatically selected by the code so that the model remains stateful.

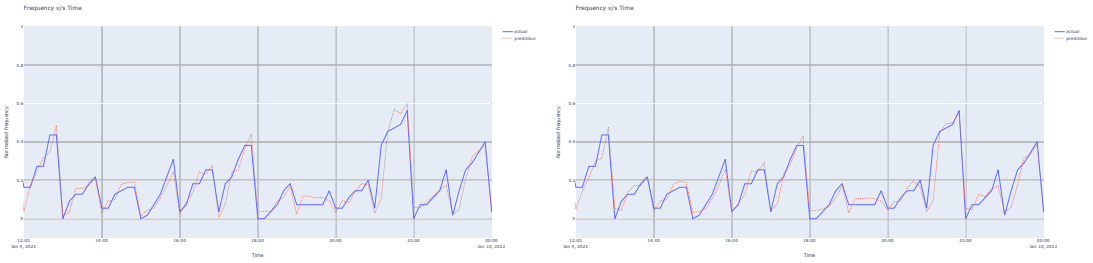
The first observation made based on data collected in Table 2 is the role of different learning rates in prediction accuracy. Regardless of the number of steps and the size of the time bucket, using a high value of learning rate (0.1), as shown in Fig. 4a the model fails to learn the pattern of events as expected. A high learning rate causes the model to take larger steps during the learning phase thus, it might converge to a suboptimal solution much quicker.



(a) Lr = 0.1, Ns=6 (b) Lr=0.001, Ns=2

Figure 4: Importance of learning rate and number of steps (time bucket = 10min)

The learning rate of 0.001, on the other hand, slows down the learning process. Thus, with the given number of epochs, the model fails to make any meaningful predictions based on the learned information. For the learning rate of 0.001, the model is seen to have very limited memory. Thus only replicates the very recent steps as shown in Fig. 4b. Although this behavior diminishes slightly as more time steps are considered, it does not improve the model to a noticeable extent. For any choice of the number of steps, a learning rate of 0.01 gives fairly good results among the three choices of learning rates.



(a) Lr=0.01, Ns=9 (b) Lr=0.01, Ns=6

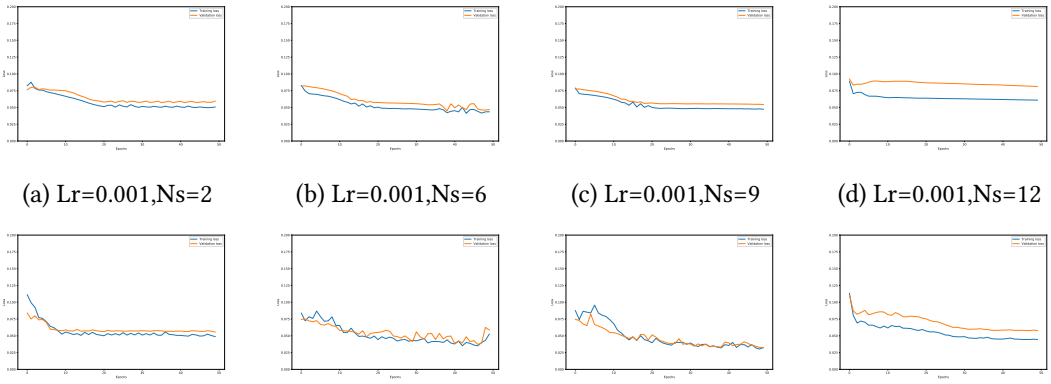
Figure 5: Impact of number of steps on predictions

Since for the majority of log analysis applications, a fast-learning model is required, in this work, the number of epochs and the learning rate are intentionally capped. Therefore, although

it is possible to further reduce the learning rate, this is not favorable as it requires a significantly larger number of epochs, which in turn extends the entire learning period.

Number of steps (Ns) also plays an equally important role in the model's accuracy. If too few or too many steps (e.g., 2 or 12) are considered, the model has almost no proper memory of the events. Using these setups the model either projects the recently seen values as its predictions, or it predicts the overall trend in data. This behavior is observed for all learning rates. The best results as shown in Figure 5a and Figure 5b are achieved in 6 and 9 steps. Although the prediction lags at certain points, in general, the model provides acceptable predictions.

Traces of routine system operations that typically occur in one-hour cycles are recorded in system logs. Therefore, observing a more accurate prediction based on the last 6 steps (60 minutes) was not unexpected. Similar observations were made for the 30-minute time bucket. Compared to the coarser time buckets, the 10-minute bucket provides better prediction accuracy. Fig. 6 illustrates an overview of the final results.



(a) Lr=0.001,Ns=2 (b) Lr=0.001,Ns=6 (c) Lr=0.001,Ns=9 (d) Lr=0.001,Ns=12

(e) Lr=0.01,Ns=2 (f) Lr=0.01,Ns=6 (g) Lr=0.01,Ns=9 (h) Lr=0.01,Ns=12

Figure 6: Variation in loss plot for 10 min bucket with changing learning rates and and number of steps

Multivariate data The multivariate setup requires different parameter combinations in comparison to the univariate setup. Therefore, similar experiments were done to evaluate the usefulness of anonymized system logs for multivariate analysis via the LSTM model. The outcome of multivariate analysis using the provided parameters is shown in Table 2.

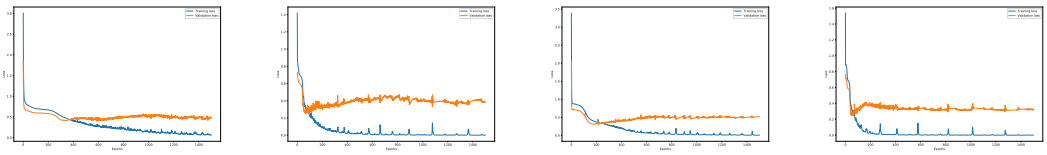
As expected, using 6-time steps provides better results compared to using only 2 time steps. This is intuitive as with 2-steps the model has limited memory and has a lagging prediction. Although the training and validation loss gets slightly better with different adjustments, none of these setups result in a sufficiently accurate model as seen for the univariate dataset.

Automatic hyperparameter optimization: To further optimize the set of hyperparameters for the model, additional testing was done on the univariate data using the Keras Tuner[10]

[10]Available at https://keras.io/keras_tuner/

library. According to Figure 5a, the bucket size of 10 minutes and 9 steps were chosen for this experiment. The same model as defined in Section 4 was employed. The learning rate was varying between 0.0001 and 0.001 and the mean squared logarithmic error (MSLE) was chosen as the loss function. It is worth mentioning that MSLE was used for the Keras tuner as using other loss functions resulted in the model getting stuck in a local minimum.

In addition, to explore the impact of model complexity, the number of LSTM layers in the proposed model was alternating between 1 and 3 layers. To assure the correctness and completeness of the observations, the number of epochs was also increased to 1500. Figure 7 provides an overview of the outcome in various setups.



(a) Lr=0.0001, 1-L (b) Lr=0.001, 1-L (c) Lr=0.0001, 3-L (d) Lr=0.001, 3-L

Figure 7: Loss plots for 10 min bucket and 9 steps over 1500 epochs.

The best model with 128 neurons and a learning rate of 0.0006 selected by the Keras tuner did not provide any significant improvement compared to the initial setup, built by manual experiments. Furthermore, additional layers of LSTM did not improve the predictions. In a single-layer setup, overfitting of the model after 100 epochs confirms the correctness of choosing less number of epochs for the provided dataset.

An interesting observation made from Figure 7b is the presence of a mirroring effect on training and validation loss. This could be an indication of noises in the monitoring data. Thus, further pre-processing steps might be necessary to improve the accuracy of models trained by anonymized system logs.

Finally, it can be concluded that univariate analysis is more effective for training LSTM models using anonymized system logs encoded via the *PαRS* anonymization method. Despite the improvements observed for varying setups, generally, the effect of training is missing which can be seen from most of the loss plots. This could be a consequence of the anonymization done on the data. However, the usefulness of this data for anomaly detection can still be confirmed from this analysis.

6. Conclusion and future works

This work provided a comprehensive assessment on the usefulness of fully anonymized monitoring data for anomaly detection using LSTM models. System logs due to their availability on current HPC systems and their information richness are desired monitoring data for behavioral analysis. In addition, conclusions derived from Syslog analysis can be generalized to a wide range of computing systems. To address the privacy concerns raised due to the existence of sensitive data in system logs, the *PαRS* anonymization mechanism is applied. *PαRS* preserves the similarity of system logs while encoding them into a stream of hashed messages. Based on

previous works, it was known that 80% of the system logs on Taurus are generated by the top 10 most frequent patterns. The normal behavior of the system is expected to be dictated by these highly frequent patterns. Therefore, the frequency of appearance of the top 10 patterns among resulting anonymized system logs, calculated within a specified time bucket, was chosen as the quantitative metric. An unsupervised machine learning model, namely LSTM was chosen. The size of the model, the amount of required data, and the number of epochs were kept to their minimum, to match the dynamic nature of HPC monitoring data. The selection of effective hyper-parameters was made rationally by considering the prior information. The prototype was implemented in Python using Keras. The python code, Syslog data, and information on how to reproduce this work are available at [12].

According to the analysis, the system logs anonymized by *PαRS*, despite the complete concealment of log information, are still usable even in the simplest LSTM models for behavioral analysis. Therefore, the usefulness of such anonymized data for anomaly detection via LSTMs is confirmed. However, the best model found after fine-tuning was seen to predict the pattern to a certain extent but not with significantly high accuracy, mainly caused by the shortcomings present in the monitoring data. In future work, more quantitative data such as power consumption and temperature variations will be added to the model, to further improve the accuracy of behavioral analysis. Significant deviation from the identified normal systems behavior could be the signal of potential anomalous behavior. However, defining the correct criteria for such a threshold is a challenging topic which in future works will be addressed. Furthermore, implementing a robust pipeline for anomaly detection is planned.

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF, 01/S18026A-F) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig". The authors gratefully acknowledge the GWK support for funding this project by providing computing time through the Center for Information Services and HPC (ZIH) at TU Dresden on HRSK-II.

References

[1] Siavash Ghiasvand, Florina M. Ciorba, Ronny Tschüter, Wolfgang E. Nagel: Lessons Learned from Spatial and Temporal Correlation of Node Failures in High Performance Computers, 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), pp. 377-381, Greece (2016)

[2] Siavash Ghiasvand, Florina M. Ciorba: Anonymization of System Logs for Preserving Privacy and Reducing Storage, Future of Information and Communication Conference (FICC), Singapore (2018)

[3] Siavash Ghiasvand: uPAD: Unsupervised Privacy-Aware Anomaly Detection in High Performance Computing Systems, Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods (ICPRAM), pp. 852-859, Czech Republic (2019)

[4] Ma, Qicheng, and Nidhi Rastogi. "DANTE: Predicting Insider Threat using LSTM on system logs." In 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1151-1156. IEEE, 2020.
[5] Du, Min, Feifei Li, Guineng Zheng, and Vivek Srikumar. "Deeplog: Anomaly detection and diagnosis from system logs through deep learning." In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp. 1285-1298. 2017.
[6] Wang, Mengying, Lele Xu, and Lili Guo. "Anomaly detection of system logs based on natural language processing and deep learning." In 2018 4th International Conference on Frontiers of Signal Processing (ICFSP), pp. 140-144. IEEE, 2018.
[7] Zhang, Ke, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. "Automated IT system failure prediction: A deep learning approach." In 2016 IEEE International Conference on Big Data (Big Data), pp. 1291-1300. IEEE, 2016.
[8] Chen, Zhuangbin, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R. Lyu. "Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection." arXiv preprint arXiv:2107.05908 (2021).
[9] Chen, Hao, Ruizhi Xiao, and Shuyuan Jin. "Unsupervised Anomaly Detection Based on System Logs."
[10] Huangfu, Yixin, Saeid Habibi, and Alan Wassyng. "System Failure Detection Using Deep Learning Models Integrating Timestamps With Nonuniform Intervals." IEEE Access 10 (2022): 17629-17640.
[11] Siavash Ghiasvand, Florina M. Ciorba: Anomaly Detection in High Performance Computers: A Vicinity Perspective, ISPDC, Netherlands (2019)
[12] Tom Richard Vargis. 2022. LSTM model on HPC Cluster monitoring data. https://github.com/tomrv22/hpcmonitorcode.git