Log Analytics with NLP

Stanford CS224N {Custom Project

Vishal Murgai
SCPD, Department of Computer Science
Stanford University
vmurgai@stanford.edu

Abstract

Cloud computing infrastructure is an integral part of our daily apps, ranging from shopping, and travelling to cellular communications. Given the pervasive use of cloud computing, keeping track of its health is of utmost importance to its providers. Large scale of deployment of this aggravates the complexity. All cloud applications generate logs which are raw text messages to reflect run-time execution. Analyzing such log messages and training a model can help cloud operator timely detect an anomaly, before it leads to service disruption. In this paper, we apply various NLP models to detect system anomalies by analyzing logs from cloud applications and evaluate their precision and accuracy of anomaly detection. We have added attention layer to LSTM and GRU models to improve accuracy. However, attention layer addition didn't yield expected results for LSTM. Log messages are typically short sentences, linguistically much simpler, and don't suffer from long-range dependencies, hence adding attention layer didn't help for LSTM. We also studied GRU models with attention and found them to be a good fit for our dataset in cloud environment. It has reduced instruction set, faster training time, hence are much more energy efficient without compromising on anomaly detection accuracy.

1 Key Information to include

- Mentor: Kaili Huang
- External Collaborators (if you have any): No
- Sharing project: No

2 Introduction

Cloud applications generate periodical logs that contain detailed run-time behavior in production networks. These logs are utilized for cloud management for quality assurance. Given the enormous scale of logs, and ever increasing cloud infrastructure, there is a need to automate the analysis for trouble-shooting or predicting impending issues. One of the challenges is that logs are mostly unstructured and often context sensitive. Analysis of such logs also needs deep domain knowledge, a flexible log parser, rich labelled dataset and a sophisticated NLP model. In this paper, we explore different NLP models, experiment and analyze the results. Classifying anomalies accurately is a critical step maintaining a healthy cloud infrastructure. The data center workload and hence log messages are collected over a period of time and in a regular sequence or pattern. This makes them suitable for RNNs that have the ability to record long time dependencies. We have explored adding attention layer to LSTM and GRU models of RNN. Our project is influenced with mainly three research papers, DeepLog [1], energy efficient data-centers [2] and comparative study of GRU vs LSTM [3], as described in Section 3. The subsequent sections elaborate our overall approach, model architecture, results and analysis. We conclude the paper with ideas on further improvements and next steps.

3 Related Work

3.1 DeepLog: Log Analytics using LSTM

The research paper [1] uses an open-source public labelled dataset for cloud applications, such as HDFS and Openstack. It uses RNN with LSTM model to identify anomalies in log data. The paper also compares the results obtained for HDFS dataset with previous research using other approaches such as PCA, TF-IDF, N-gram etc. The paper has researched and proto-typed a working LSTM model for two mainstream cloud applications. However, the paper has explored other variants of RNNs (such as attention to LSTM, GRUs, Transformer etc.). The paper doesn't look into potential anomalies caused due to interaction of different cloud applications. Lack of labelled dataset could be one of limitations the authors may have faced.
Research in anomaly detection algorithms has been an active areas in many decades. But with recent advances in machine learning and natural language processing, the task for anomaly detection can be done with higher accuracy than ever before. Broadly the approaches for anomaly detection can be divided in three categories [4]:

1. Deep Learning for feature extraction: These techniques uses deep learning architectures to extract feature vector and then detect anomaly on the basis of this extracted vector. This category of methods aims at leveraging deep learning to extract low-dimensional feature representations from high-dimensional and/or non-linearly separable data for downstream anomaly detection. The feature extraction and the anomaly scoring are fully disjointed and independent from each other. Thus, the deep learning components work purely as dimensionality reduction only. One research line is to directly uses popular pre-trained deep learning models, such as AlexNet[5] or VGG [6], to extract low-dimensional features
2. Learning Feature Representations of Normality: The methods in this category couple feature learning with anomaly scoring in some ways, rather than fully decoupling these two modules as in the last section. These methods generally fall into two groups: generic feature learning and anomaly measure-dependent feature learning. Generic normality feature learning methods like autoencoders [7] learns the representations of data instances by optimizing a generic feature learning objective function that is not primarily designed for anomaly detection, but the learned representations can still empower the anomaly detection, since they are forced to capture some key underlying data regularities.
3. End-to-end Anomaly Score Learning:This category aims at learning feature representations that are specifically optimized for one particular existing anomaly measure. The advantages of this category of methods are as follows. The distance-based anomalies are straightforward and well defined with rich theoretical supports in the literature. Thus, deep distance-based anomaly detection methods can be well grounded due to the strong foundation built in previous relevant work. They work in low-dimensional representation spaces and can effectively deal with high-dimensional data that traditional distance-based anomaly measures fail. They are able to learn representations specifically tailored for themselves.

Our project is based on research paper [1] and open-source implementation [8]. We have developed our own baseline using tensorflow framework for LSTM, and then added attention layer to LSTM and GRU models.

3.2 Energy efficient GRUs for data center

The research paper [2] aims at reducing energy consumption by servers in cloud infrastructure by moving and consolidating VMs into a single physical server. Selection of which VMs to move is an important challenge. The researchers have classified traffic pattern to a VM as latency sensitive and insensitive. The classification is done using CNN and GRU based model [2].
The research paper [3] analyzes and compares GRUs and LSTM's performance on Yelp dataset. The paper finds that training with GRU is 29.29% faster than LSTM for same dataset.

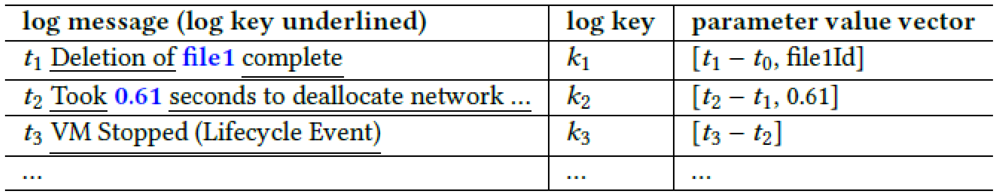
In our project, we have explored adding attention layer to LSTM and GRU to improve accuracy. We compare training time for the two models and evaluate their performance. We implemented energy efficient and cost effective GRUs [2]. We would describe our approach, model, and results in subsequent sections.

4 Approach

4.1 Log Analysis Architecture

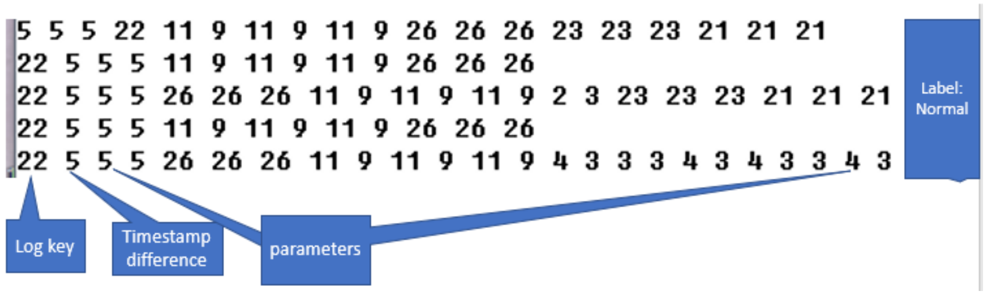
In this subsection, we would briefly lay out the overall architecture of log analysis, different stages of pipeline and associated functionality. The Figure 3 shows overall architecture pipeline. Log collection and Parameter Value Vector generation are leveraged from open-source [8].

Log collection: Logs consists of different sequence of events generated at runtime by execution of structured source code. Figure 1 shows an example of this. We are using pre-collected and pre-processed logs as shown in figure 3 from Pytorch implementation [8].



log message (log key underlined)	log key	parameter value vector
t_1 Deletion of file1 complete	k_1	$[t_1 - t_0$, file1Id]
t_2 Took 0.61 seconds to deallocate network ...	k_2	$[t_2 - t_1, 0.61]$
t_3 VM Stopped (Lifecycle Event)	k_3	$[t_3 - t_2]$
...	...	...

Figure 1: Log Example. Figure taken by original paper[1]



```
5 5 5 22 11 9 11 9 11 9 26 26 26 23 23 23 21 21 21
22 5 5 5 11 9 11 9 11 9 26 26 26
22 5 5 5 26 26 26 11 9 11 9 11 9 2 3 23 23 23 21 21 21
22 5 5 5 11 9 11 9 11 9 26 26 26
22 5 5 5 26 26 26 11 9 11 9 11 9 4 3 3 3 4 3 4 3 3 4 3
```

Figure 2: Example: Parameter Vector Value for "Normal" sequence

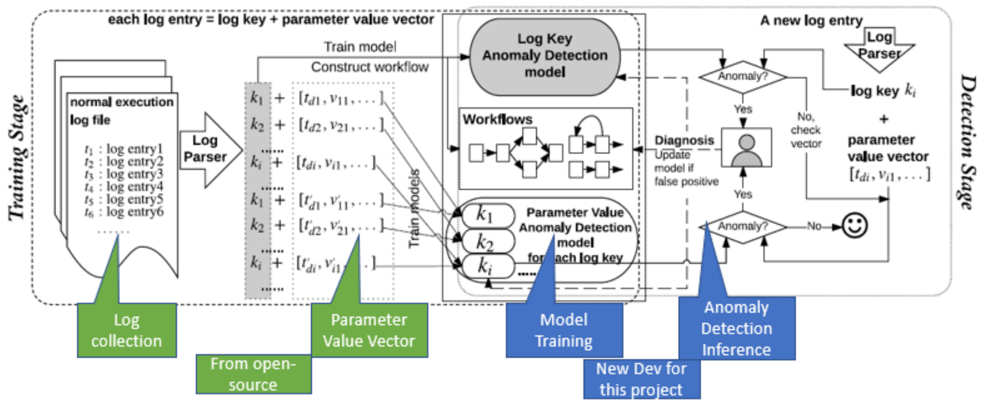

each log entry = log key + parameter value vector
Training Stage
normal execution log file
t_1 : log entry1
t_2 : log entry2
t_3 : log entry3
t_4 : log entry4
t_5 : log entry5
t_6 : log entry6
Log Parser
Train model
Construct workflow
k_1 + $[t_{d1}, v_{11}, \dots]$
k_2 + $[t_{d2}, v_{21}, \dots]$
k_i + $[t_{di}, v_{i1}, \dots]$
k_1 + $[t'_{d1}, v'_{11}, \dots]$
k_2 + $[t'_{d2}, v'_{21}, \dots]$
k_i + $[t'_{di}, v'_{i1}, \dots]$
Train models
Log Key Anomaly Detection model
Workflows
Parameter Value Anomaly Detection model for each log key
Diagnosis
Update model if false positive
A new log entry
Log Parser
Anomaly?
Yes
No, check vector
log key k_i
+
parameter value vector
$[t_{di}, v_{i1}, \dots]$
Anomaly?
No
Detection Stage
Log collection
From open-source
Parameter Value Vector
Model Training
New Dev for this project
Anomaly Detection Inference

Figure 3: Log Analysis Architecture. Figure taken by original paper [1]

Log parsing: The objective of log parsing is to transform unstructured log messages into a vector map of numbers, so that NLP methods could be applied on those for analysis. For this project, we are using pre-processed dataset published in the Pytorch implementation [8].

From text logs to Parameter Vector Value: An example log entry is shown in Figure 1. Each log entry has a log key or module identifier that identifies message type. For example, in the first row in Figure 1, log key would be "Deletion of file complete". t_1, t_2, t_3 are timestamps of consecutive log messages. In the parameter value vector, we take timestamp difference between two consecutive messages as the first element, which is [t_1 - t_0], file identifier as second element. In the third row of Figure 1, "VM stopped" is the log key and there is no file identifier associated, so the parameter vector is just difference of timestamps [t_3-t_2]. Sequence of logs are labelled as normal or abnormal based on annotations done by HDFS experts. The figure 2 shows an example of vectorized parameter value for **normal** label. There are a total of 28 classes in the dataset.

Model: We developed three different models from scratch as part of this project: Tensorflow LSTM to set a baseline, TF LSTM with attention, and GRU with attention. Each log key sequence parsed from training logset (shown in figure 2) is used by our model to train anomaly detection model. It builds sort of system run-time execution mental model for triage and debugging. For each unique key k, the model also trains and builds a model for detecting system performance anomalies by using parameter value vector sequence of log key, k. The next sub-section describes the models in detail.

Anomaly detection: Anomaly detection models are trained to check whether a newly arrived log is normal. It compares log key is valid and then evaluates the parameter value vector for normal or abnormal behavior as shown in Figure 3.

4.2 Model Architecture

We implemented a tensorflow LSTM model to set a baseline for us. This model is shown in Figure 4. We further added an attention layer to LSTM as shown in Figure 5. We also developed GRU with attention as in Figure 6.

The RNN variants such as LSTM and GRU suffer from long-range dependency problem. We added an attention layer to help alleviate that. The improved model Figure 6 consists of one GRU layer, one attention layer, and one dense output layer. The size of the output layer is equivalent to the number of classes in the dataset.

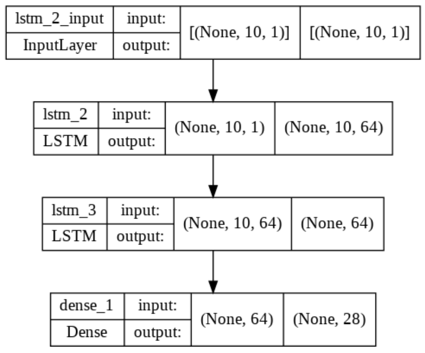

lstm_2_input | InputLayer | input: | output: | [(None, 10, 1)] | [(None, 10, 1)]
lstm_2 | LSTM | input: | output: | (None, 10, 1) | (None, 10, 64)
lstm_3 | LSTM | input: | output: | (None, 10, 64) | (None, 64)
dense_1 | Dense | input: | output: | (None, 64) | (None, 28)

Figure 4: LSTM Model

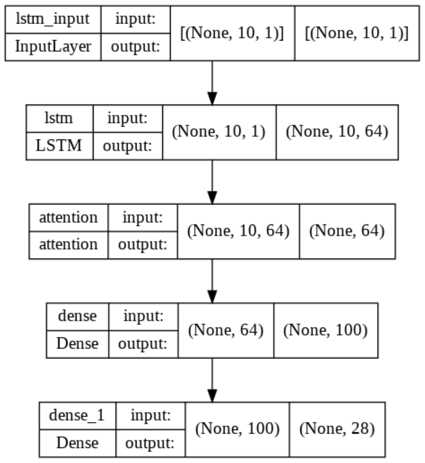

lstm_input | InputLayer | input: | output: | [(None, 10, 1)] | [(None, 10, 1)]
lstm | LSTM | input: | output: | (None, 10, 1) | (None, 10, 64)
attention | attention | input: | output: | (None, 10, 64) | (None, 64)
dense | Dense | input: | output: | (None, 64) | (None, 100)
dense_1 | Dense | input: | output: | (None, 100) | (None, 28)

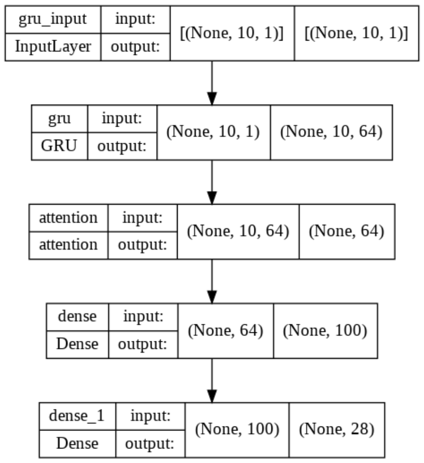
Figure 5: LSTM with Attention

We have explored two RNN variations, LSTM and GRU. Both of these algorithms were proposed to control the memorization process in traditional RNN. We have chosen the GRU as the underlying unit GRU is less complex than LSTM, and GRU uses fewer tensor operations and is significantly faster than LSTM. Thus, GRU takes less time to train, and consumes less system resources, and more energy efficient.

The Hidden State h_t of the GRU is computed as follows [9]:
$z_t = \sigma(G_{xz}x_t + G_{hz}h_{t-1})$.
$r_t = \sigma(G_{xr}x_t + G_{hr}h_{t-1})$.
$\tilde{h}_t = \tanh(G_{xh}x_t + G_{rh}(h_{t-1} \otimes r_t))$.

$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h}_t$.
where $\sigma(\cdot)$ denotes the sigmoid function, and $\otimes$ denote the element-wise multiplication operator. The update gate, the candidate activation, and the reset gate are represented by z_t, $\tilde{h}_t$ and r_t respectively. The connections from the input to the hidden nodes are parameterized by matrices G_{xz}, G_{xr}, and G_{xh}. Connections from the hidden to the output nodes are parametrized by matrices G_{hz}, G_{hr}, and G_{rh}. The function z_t is the filter that is used for mapping the previous state. The function x_t that represents the current state does not influence the output state if the z_t function is low; x_t influences the output at the current state only if the z_t function is high.

GRU includes reset and update gates that can help to overcome gradient vanishing and exploding problems in RNNs and it also has a fewer number of training parameters compared to LSTM. Reset and update gates in GRU are two vectors that create a very precise control mechanism using the Sigmoid activation function and decide how much of the previous memory is related to the current input. The architecture of GRU cell is depicted in Figure 7.



gru_input | InputLayer | input: | output: | [(None, 10, 1)] | [(None, 10, 1)]
gru | GRU | input: | output: | (None, 10, 1) | (None, 10, 64)
attention | attention | input: | output: | (None, 10, 64) | (None, 64)
dense | Dense | input: | output: | (None, 64) | (None, 100)
dense_1 | Dense | input: | output: | (None, 100) | (None, 28)

Figure 6: GRU Model with Attention

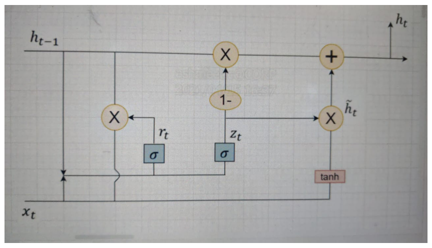


Figure 7: GRU cell

5 Experiments

5.1 Data

We are using labeled dataset from PyTorch LSTM implementation [8], which is inspired by the research paper [1]. It provides pre-parsed and vectorized parameter values for training and test log dataset for HDFS application [10]. The test set has log sequences for normal and abnormal events. The figure 1 show an example of log messages and Figure 2 shows an example of vectorized parameter value. The Figure 3 shows overall pipeline stages. We feed the training log dataset represented by parameter vector to tensorflow based LSTM model for training. The test dataset is fed into the model for detecting anomalies as shown in the figure 3. There are 28 classes, similar to [1]. The following table 1 shows dataset specification.

Type	Training	Testing
normal	4,855	553,366
abnormal	1,638	15,200

Table 1: HDFS Dataset Details

5.2 Evaluation method

We have used F1-Score, precision and recall as a metric for evaluation of our model. We have also used number of false positive (FP) and false negative (FN) to evaluate our model.

5.3 Experimental details

We have used Microsoft Azure with 6 CPUs and a single 12GB Nvidia K80 GPU for our training, testing and validation. Our model is trained so as to minimize the binary cross entropy loss function given by

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i)) \tag{1}$$

where y_i is the label/target value/binary indicator (0 or 1), and $p(y_i)$ is the predicted probability of an observation (i.e. the value in the model output) being 0 or 1. The loss is calculated for every epoch.

For the GRU model, We attempted a few hyper-parameter tuning, such as batch size, number of epochs, and found that the values captured in 2 provide the best results. For the LSTM models, We tried to add another LSTM layer to the model in stacked LSTM architecture, but it didn't help in improving the accuracy. It did increase the training time.

Table 2: Simulation parameters of GRU attention

Parameter	**Value**
Output size $\|X_d\|$	28
Weight initializer	He normal
Learning rate η	10^{-4}
Loss function $\mathcal{L}$	Binary cross entropy
Epochs n	100
Optimizer	ADAM [11]
Batch size	128

5.4 Results

This section describes the results of our experiments. The first row shows the results of a PyTorch LSTM implementation [8]. We developed our own tensorflow LSTM model and the second row shows the results for various evaluation metrics we have used. We added an attention layer to LSTM and also implemented GRU based model as reflected in the tables. As shown in table 3, we observed that adding Attention layer to LSTM didn't yield expected results. However, adding attention layer to GRU yielded slightly better performance (F1-Score) as compared to LSTM with attention. We also compared the training time taken by LSTM vs GRU attention models and observed that GRU is faster as compared to LSTM as shown in Table 5. The research paper [3] also echoes the same behavior.

False Positive	False Negative	Model
566	314	DeepLog PyTorch LSTM [8]**Baseline**
650	173	CS224n Orig Tensorflow LSTM
796	218	CS224n TF LSTM with attention
708	207	CS224n TF GRU with attention

Table 3: Results: FP and FN for different NLP models

Precision	Recall	F1-Score	Model
0.87	0.92	0.89	DeepLog PyTorch LSTM [8] **Baseline**
0.85	0.95	0.90	CS224n Orig Tensorflow LSTM
0.83	0.95	0.88	CS224n TF LSTM with attention
0.83	0.94	0.89	CS224n TF GRU with attention

Table 4: Results: Precision, recall and F1-score for different NLP models

The number of False Positive with GRU attention improved as compared to LSTM with attention, but overall FP didn't improve from the Pytorch or TF baselines. But False Negative, which is critical metric improved with GRU attention, as compared to LSTM attention. We also noticed that overall

Training Time (min)	Model
190	CS224n Orig Tensorflow LSTM with attention
175	CS224n TF GRU with attention

Table 5: Results: Training time comparison

tensorflow based model outperformed PyTorch based baseline model for false negatives. On the other hand, False positive also increased with TF implementations as compared to PyTorch.

6 Analysis

Self-attention works well at finding relations between words or items independent of how far apart they appear in the sequence. LSTM is also good at this functionality as well. So, adding attention to LSTM doesn't help much as they practically do the same thing. Adding attention only adds to the noise.
The long range sentence issue is a not relevant in the context of log messages. These logs are much linguistically simpler and typically short sentences as shown in Figure 1. We could have added a pre-processing layer to identify bag of words, their weights and pass it to LSTM and GRU attention layers respectively. This would extract and capture the essential keywords from logs to help improve accuracy. Another possible reason could be quality of dataset itself. Unfortunately, since we don't have a log parser, we can't check the quality of raw logs in the open-source dataset [1].

We found GRU to be very cost effective [2]. GRU has two gates that are reset and update while LSTM has three gates that are input, output, forget. This reduces the complexity of the model, takes much less training time, and provides almost similar accuracy as compared to LSTM. F1-score for GRU attention model is .89 as compared to .90 of highly complex LSTM model. LSTM with attention took 190 minutes to train as compared to GRU with attention model that took only 175 minutes. Energy efficient is a critical benchmark for data centers and helps reduce carbon footprint for a better sustainable environment.

It is also interesting to note the difference in performance between PyTorch LSTM (first row) and Tensorflow LSTM (second row) for almost similar configuration and parameters. We would like to root cause this further to get deeper perspective.

7 Conclusion

Accurate prediction of anomalies is key to smooth and uninterrupted operation of cloud compute services. In this paper, we analyzed LSTM, LSTM with attention, and GRU with attention variants of RNN. Adding attention to LSTM didn't yield promising results, potentially due to simplistic nature of language sentences in log messages, which don't suffer from long-range sentence issues in linguistic literature. We found GRU with attention to be a good candidate for deployment in the cloud for anomaly detection. GRU is less complex as compared to LSTM, faster training time, more energy efficient, equally accurate and very cost effective for anomaly detection in cloud infrastructure.

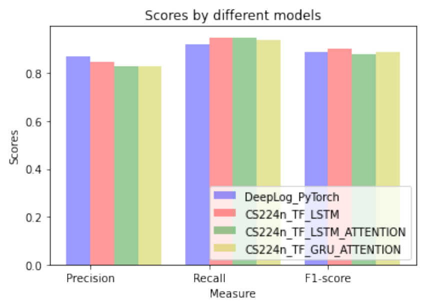


Figure 8: Scores for different models

In future, we would like to get a working parser to evaluate various raw log messages, and study the behavior of multiple cloud applications. We would also like to add a pre-processor stage with bag-of-words to extract keywords and weights, and pass it to the LSTM/GRU models. This would allow us to customize processing based on keywords often appearing in application specific logs and passing useful hints for attention. We would also like to add CNN layer with LSTM/GRU and see how it helps with improving accuracy.

References

[1] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[2] Zeinab Khodaverdian, Hossein Sadr, S. A. Edalatpanah, and Mojdeh Nazari Solimandarabi. Combination of convolutional neural network and gated recurrent unit for energy aware resource allocation. *ArXiv*, abs/2106.12178, 2021.

[3] Shudong Yang, Xueying Yu, and Ying Zhou. Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example. In *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, pages 98–101, 2020.

[4] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR)*, 54(2):1–38, 2021.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[8] Yifan Wu. Pytorch Implementation of DeepLog. `https://github.com/wuyifan18/DeepLog`, 2019.

[9] Jianshu Zhang, Jun Du, and Lirong Dai. A gru-based encoder-decoder approach with attention for online handwritten mathematical expression recognition. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 902–907, 2017.

[10] Open-source. HDFS Architecture Guide. `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`, 2000.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR*, 2015.

A Appendix

We would like to sincerely thank our mentor Kaili Huang for guidance on this project. We would also like to thank the entire teaching staff for making this course an invaluable experience.