View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by SJSU ScholarWorks

San Jose State University
SJSU ScholarWorks

Master's Projects

Master's Theses and Graduate Research

Spring 5-22-2019

Intelligent Log Analysis for Anomaly Detection

Steven Yen
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Artificial Intelligence and Robotics Commons, and the Information Security Commons

Recommended Citation

Yen, Steven, "Intelligent Log Analysis for Anomaly Detection" (2019). *Master's Projects*. 739.
DOI: https://doi.org/10.31979/etd.h4j5-8ctj
https://scholarworks.sjsu.edu/etd_projects/739

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Intelligent Log Analysis for Anomaly Detection

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Masters of Science

By

Steven Yen

May 2019

The Designated Project Committee Approves the Project Titled

Intelligent Log Analysis for Anomaly Detection

By

Steven Yen

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

San Jose State University

May 2019

Dr. Melody Moh	Department of Computer Science
Prof. Auston Davis	Department of Computer Science
Dr. Katerina Potika	Department of Computer Science

ABSTRACT

Intelligent Log Analysis for Anomaly Detection

By Steven Yen

Computer logs are a rich source of information that can be analyzed to detect various issues. The large volumes of logs limit the effectiveness of manual approaches to log analysis. The earliest automated log analysis tools take a rule-based approach, which can only detect known issues with existing rules. On the other hand, anomaly detection approaches can detect new or unknown issues. This is achieved by looking for unusual behavior different from the norm, often utilizing machine learning (ML) or deep learning (DL) models. In this project, we evaluated various ML and DL techniques used for log anomaly detection. We propose a hybrid neural network (NN) we call "CausalConvLSTM" for modeling log sequences, which takes advantage of both Convolutional Neural Network and Long Short-Term Memory Network's strengths. Furthermore, we evaluated and proposed a concrete strategy for retraining NN anomaly detection models to maintain a low false-positive rate in a drifting environment.

ACKNOWLEDGEMENTS

This master's project has been a substantial undertaking for me, and I would like to thank my project advisor Dr. Melody Moh for all her support and guidance through the entire process, from my initial project selection, to development, to completion. I would like to thank Dr. Teng Moh for all the feedback and insights he has provided throughout the development of the project. I would like to thank Professor Auston Davis and Dr. Katerina Potika for serving as my committee members, and spending time to discuss the project with me and sharing ideas and other considerations. I would like to thank all the CS department faculty members and my friends at SJSU for making my pursuit of a master's degree more fulfilling and pleasant. Finally, I would like to thank my family for always being there for me.

TABLE OF CONTENTS

1 - Introduction 9
2 - Background and Related Works 11
2.1 Preliminaries 11
2.2 Anomaly Detection 13
2.3 Concept Drift 17
3 - Problem Statement 19
4 - Proposed Solution 19
4.1 Data Preprocessing Module 20
4.2 Core Anomaly Detection Model 21
4.3 Online Update and Retraining Module 25
5 - Performance Evaluation 26
5.1 Anomaly Detection Evaluation 27
5.2 Online Update and Retraining Evaluation 31
6 – Discussion and Lessons Learned 36
6.1 Alternate Approaches (Non-Sequence Based) 36
6.2 Lessons Learned 43
7 - Conclusion & Future Work 45
LIST OF REFERENCES 46

LIST OF FIGURES

Fig. 1. Typical ML/DL Log Anomaly Detection Pipeline 11

Fig. 2. Example HDFS Log 12

Fig. 3. Taxonomy of Anomaly Detection Techniques in Recent Literature 13

Fig. 4. Proposed CausalConvLSTM Architecture 23

Fig. 5. Causal Convolution as Used in Our Architecture 24

Fig. 6. Model Accuracies on HDFS Dataset 28

Fig. 7. Histogram of MSEs from Validation Set (CICIDS2017, CausalConvLSTM Model) 30

Fig. 8. GRU vs CausalConvLSTM on CICIDS2017 Dataset 31

Fig. 9. Overall FP Rate on Entire BGL Test Set with Different Retraining Strategies (GRU AD Model) 32

Fig. 10. GRU Model Metrics over Time, with Retraining Based on Rolling Window FP Rate 35

Fig. 11. K-Means Accuracy vs Target Number of Clusters (HDFS Dataset) 38

Fig. 12. Clustering, Reconstruction, and Sequence-Based AD Comparison (HDFS Dataset) 40

Fig. 13. Clustering, Reconstruction, and Sequence-Based AD Comparison (CICIDS17 Dataset) 42

LIST OF TABLES

Table 1. Model Description and Performance Results on HDFS Dataset 28

Table 2. Model Performance Results on CICIDS2017 Dataset 30

LIST OF ACRONYMS

- AD – Anomaly Detection
- AE – Autoencoder
- ANN – Artificial Neural Network
- BGL – Blue Gene/L Supercomputer
- CNN – Convolutional Neural Network
- DL – Deep Learning
- GRU – Gated Recurrent Unit
- HDFS – Hadoop Distributed File System
- LSTM – Long Short-Term Memory
- ML – Machine Learning
- MLP – Multi-Layer Perceptron
- NLP – Natural Language Processing
- PCA – Principal Component Analysis
- RNN – Recurrent Neural Network

1 - Introduction

Computer systems generate large volumes of logs recording various events of interest. These can be generated by various sources including operating systems, network devices, applications, and so on. These logs are used by developers and system administrators for troubleshooting, auditing, and identifying various issues.

As computing systems become more complex and the scale grows, the volumes of logs generated also increase tremendously. The manual analysis of logs using search utilities become ineffective, as it is often hard for humans to correlate events across large volumes of logs across time and many different systems. Additionally, manually analysis by humans is often not timely enough to keep up with the large throughput of events. This resulted in the development of automated log analysis tools, the earliest of which are rule-based systems. These systems rely on a set of rules (in a rule-set) written by experts, defining the patterns of interest. A rule-engine then applies those rules to new, incoming events to detect if there's a match and take the appropriate actions [1]. These systems are efficient and can quickly identify matches. Unfortunately, they can only detect known issues for which a rule exists in the rule-set. These types of systems cannot detect new or unknown issues for which there are no existing rules.

To address the shortfall of rule-based systems, anomaly-based systems were developed, which leverage statistical, machine learning (ML), or deep learning (DL) techniques to mine large volumes of data to extract insights and learn their own evaluation criteria. Of these, deep learning (DL) techniques utilizing deep Artificial Neural Networks (ANN) with many layers showed the most promising results. However, these models require long training time before they can be used for detection. To improve the training time while maintaining good accuracy, we propose a hybrid model that uses a Convolutional Neural Network (CNN) in conjunction with a Long Short-Term Memory (LSTM) network. In our proposed model called "CausalConvLSTM", we first use causal convolutions with variable-sized filters to extract features based on different sequence lengths to construct a rich feature map that preserves ordering significance. Then, we use

a LSTM network to ingest the resulting feature map in order to summarize the result into a hidden state vector that is then passed to a fully-connected (FC) layer to make predictions. Based on our experiments, the hybrid architecture achieves competitive accuracies while significantly reducing the training time.

Another major challenge anomaly detection systems face is concept drift, which is the change in normal system behavior over time. Concept drift is an especially important issue in computing systems, which can change significantly over time for various reasons such as increase/decrease in the number of users, hardware changes (upgrade/downgrade), or software changes (introduction of new features). The result of this is that an anomaly detection system that is trained on normal system behavior from a baseline period will see a reduction in accuracy, as the normal system behavior becomes increasingly different form its behavior during the baseline period. To address this, we evaluated various strategies for updating anomaly detection models to adapt to concept drift. In the end, we propose a concrete strategy for retraining DL models based on false-positive rates calculated from a rolling window.

This paper is organized as follows. Section 2 will provide additional background and present related works. Section 3 will describe the problem of log anomaly detection we're addressing. Section 4 will describe our proposed solution including the refined architecture and the retraining strategy. Section 5 describes our experimental setup and the results. Section 6 includes general discussions, alternate approaches we experimented with, and lessons learned. Finally, Section 7 concludes the paper and describes future works.

2 - Background and Related Works

2.1 Preliminaries

In this project we focus on ML/DL techniques used for log anomaly detection, as they can identify new and unknown issues effectively.

Before ML/DL techniques can be used, however, the raw logs must be processed into an appropriate format, based on the technique. The typical pipeline for using ML/DL for log anomaly detection is shown in Fig. 1 below.

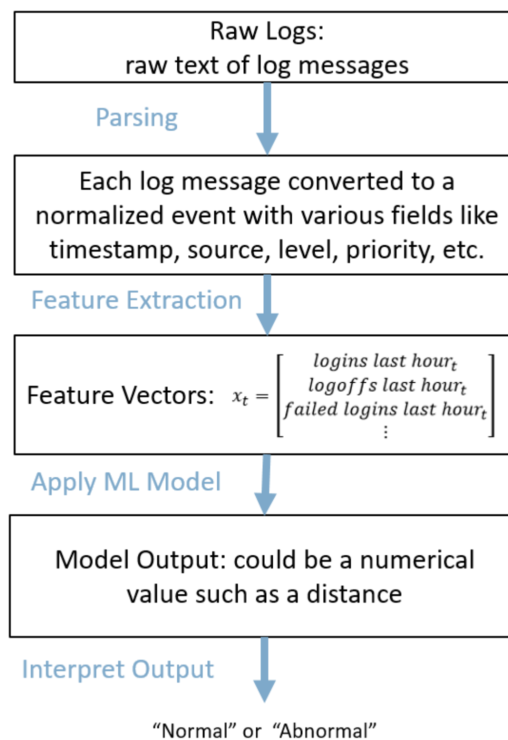

Fig. 1. Typical ML/DL Log Anomaly Detection Pipeline

First, the raw log messages are parsed to extract various fields of interest. These could include standard fields like timestamp, source, and log level that are common in most logs. However, we might also be interested in fields that are unique to specific logs or messages. For example, from the Hadoop Distributed File System (HDFS) raw log show in Fig. 2, the standard fields that we can extract include the date "081109", time "203616", PID "161", log level "INFO", and component

“dfs.DataNode$PacketResponder”. However, the remainder of this message “Received block blk_4566277459864535342 of size 67108864 from /10.251.106.50” is the free text part that can vary depending on the source code that produced the message [2]. To extract fields from this portion, we need a template that describes which portion of the message are fixed string constants, and which portion are variables. Such a “template” could be obtained through source code analysis [2]. The source code that generated the message in this example might be `printf(“Received block %s of size %d from %s”, block_id, size, srcip)`, which would give us the template “Received block (.*) of size (.*) from (.*)”, which we can use as a regular expression to parse the log and extract the values of the fields block_id, size, and srcip. The “template” is also referred to as message type [9], log key [8], or event type [3] by various researchers.

```
081109 203616 161 INFO dfs.DataNode$PacketResponder:
Received block blk_4566277459864535342 of size 67108864 from
/10.251.106.50
```

Fig. 2. Example HDFS Log

The fields extracted through such parsing could be used directly as input to ML/DL techniques as categorical (discrete) or numerical features. However, these fields are often insufficient, as they all correspond to a single event in time, and provide no information about the context. To capture higher level, contextual information, additional feature extraction is often performed. This is commonly achieved by grouping events together through time windows or common identifiers (e.g. block id) [3]. Then, for each group, we calculate various statistics such as count, min/max, average, etc. of various quantities. For example, in [3], the time is divided into intervals, and within each interval they count the number of each type of events to construct an event count vector. In [2], such an event count vector is constructed for each block id. These event count vectors are then used as feature vectors for various ML/DL techniques.

2.2 Anomaly Detection

The ML/DL techniques used for anomaly detection can be broadly divided into supervised, unsupervised, and semi-supervised [3], based on what kind of data is required for training. Fig. 3 below shows a broad categorization of anomaly detection techniques based on our literature review. Note, this is not a comprehensive list of all the possible anomaly detection techniques, but simply those we encountered in our application domain.

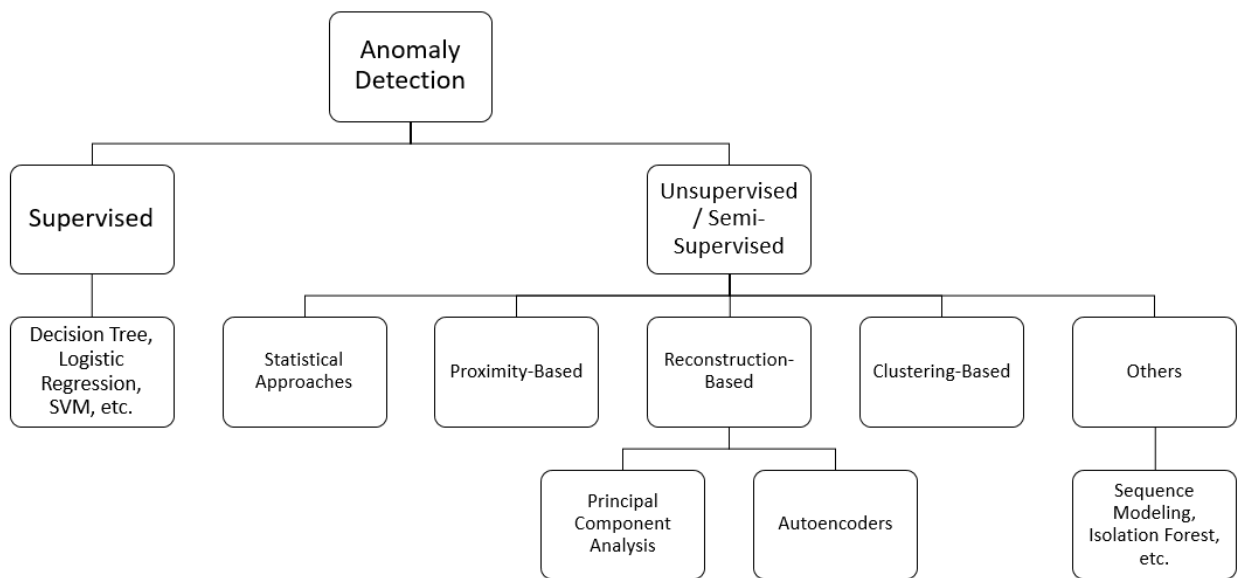

Fig. 3. Taxonomy of Anomaly Detection Techniques in Recent Literature

There has been previous works using supervised approaches for anomaly detection involving the use of Logistic Regression, Decision Trees, or Neural Networks as binary classifiers to directly classify behaviors as normal or abnormal [3][4]. However, these techniques require both normal and abnormal data points during training, which is often not a very practical requirement. Oftentimes, we won't have labeled anomaly data, as anomalies are by definition rare. Furthermore, even if we do have labeled anomaly data, a model trained on such a dataset would only be able to detect types of abnormal activities represented in the training set, and not new or unseen abnormal activities.

Due to the large volume of logs generated by computers, we often do not have labeled data, as labeling would be labor-intensive or infeasible. As such, unsupervised methods that require no labels at all, or semi-supervised methods that only require data from the normal class [11], are much more practical. Many authors do not make the distinction between unsupervised and semi-supervised anomaly detection methods, and simply refer to models that require only normal (benign) data for training as unsupervised [13]. This is understandable, since there is only one class label (i.e. "normal") in the training data, and the class label is not explicitly used by the ML/DL algorithms. Unsupervised/semi-supervised anomaly detection models can be categorized into Statistical Modeling, Proximity-Based, Reconstruction-Based, Clustering-Based, and others, as shown in Fig. 3.

In statistical modeling based anomaly detection, one constructs a parametric or non-parametric model describing the distribution of data in the normal class [14]. Then, various statistics are used to evaluate new, test points to see if they are significantly different from the normal class, and if they should be deemed abnormal. Parametric models assume the normal class distribution follows some well-known family of distribution (such as Gaussian) and uses relevant statistics (such as means and standard deviation) to determine the probability that a certain data point came from that distribution. Non-parametric models do not assume the data distribution follows some known distribution, but rather use frequency of baseline observations to assign anomaly scores to new, test points. The weakness of statistical modeling based approaches is that they are not able to handle problems with many independent variables (high dimensionality) [14][15].

Proximity-based anomaly detection is based on the assumption that normal data points have similar feature values and therefore lie close to one another in the feature space, while anomalous points have significantly different feature values and hence lie far away. Proximity-based techniques can be further divided into distance-based and density-based depending on the measure used for evaluation. In distance-based technique, one typically calculate the distance from a data point to its nearest neighbors and compare that to some threshold for anomaly detection. In density-based approach, one calculates a local density for

each point based on the number of its neighbors that lie within some distance. The weakness of proximity-based approaches is that they require the calculation and comparison of the distance between each pair of data points, leading to a $O(n^2)$ run time for n number of data points [14]. Furthermore, as the number of dimensions increase, data points become extremely sparse, and distances become less meaningful [14]. The authors of [13] applied a density-based approach using Local Outlier Factor (LOF) to the detection of anomalies in Internet-of-Things (IoT) traffic, and found the technique to perform extremely poorly compared to other approaches.

Clustering based approaches are based on the assumption that there exists some underlying structure within a dataset that allows us to group them into subgroups or clusters. The most popular clustering technique used is K-Means, where the user specifies a k value representing the targeted number of clusters. Then, through an iterative process the algorithm assigns the data points to one of k clusters and identify the centroid of each cluster. Clustering based techniques are more efficient than proximity based techniques, with a runtime linear or near-linear in the number of data points [14]. Clustering have been widely used for anomaly detection in computer systems with some level of success. In [5], K-Means was used in a semi-supervised setting, where clustering is performed on normal data from some baseline period to identify normal behaviors, represented by baseline clusters and their centroids. During the detection phase, traffic that deviates significantly from the baseline (measured by distance from the closest centroid) are deemed as abnormal. The downside with the K-Means approach is that the selection of the parameter k, which represents the targeted number of clusters, has significant impact on the accuracy, and it is not always apparent what a suitable value is [6].

Reconstruction-based approaches are based on the assumption that the normal data points have a low effective dimension, meaning it can be represented by a set of derived features that is smaller than the set of original features used for its representation [14]. These techniques involve the use of feature reduction techniques such as Principal Component Analysis (PCA) or neural network Autoencoders (AE), and the appropriate interpretation of the model outputs for anomaly detection.

In Principal Component Analysis (PCA), a set of principal components that are linear combinations of the original features are derived based on the variance of the data. For use in anomaly detection, principal components along which there are high data variation are referred to as normal subspaces (or normal directions), while principal components along which there are low data variation are referred to as abnormal subspaces (or abnormal directions). Anomaly detection can be done by looking for isolated points along the abnormal subspaces that are distant from the rest of the data points [2][7]. Quantitatively, this is done by calculating the projection distance of data points onto the abnormal subspaces, and comparing that to a threshold. The intuition is that an isolated, distant point along the abnormal subspace is exhibiting a correlation that is not exhibited by the majority of the points, so that point is likely to be anomalous. The authors of [2] used PCA for log anomaly detection in Hadoop File System (HDFS) logs, and found it to be effective in detecting many types of execution anomalies. One limitation of PCA is that they only capture linear relationships between the original features; they cannot capture more complex, non-linear relationships that exist in many systems [14].

In contrast to PCA, neural network autoencoders (AE) can capture non-linear relationships. This is possible because neural networks are able to model non-linear relationships through a series of linear and non-linear transformations, made possible by the application of various activation functions. Autoencoder is a special type of neural network that consists of an encoder network that takes in the input and transforms it into some latent representation (i.e. derived features). Then, a decoder network is used to reconstruct the original input based on that latent representation. The latent representation is typically of a dimension significantly lower than the input dimension. This setup, known as undercomplete, forces the network to learn the most important relationships between the original input features, and transform them into a few derived features that retain all the most important information. The decoder, which will be trained together with the encoder, will learn to create a reconstruction of the original input based on the low-dimensional representation. The error between the model's reconstruction and the original input (known as the reconstruction error), is used to update the model weights during training [16]. To use an autoencoder for

anomaly detection, we train the network on data points from the normal class such that it can produce reconstructions of normal inputs with minimal reconstruction losses. Then, in the detection stage, new samples are fed into the model to create a reconstruction. If the reconstruction error exceeds some threshold, the sample is deemed anomalous, otherwise it is deemed normal. The authors of [17] applied a deep autoencoder to IoT traffic, and showed that it was effective in detecting Botnet attacks.

Some recent work for log anomaly detection takes inspiration from natural language processing (NLP), by modeling systems logs in a way similar to a natural language sequences [8][9]. These are based on the idea that logs are similar to natural language in that they have a certain “vocabulary” governed by the number of unique log keys in the source code, and follow certain “grammar” rules due to program execution flow. The authors of [9] used a Markov chain and the authors of [8] used a Long Short-Term Memory (LSTM) neural network to model system logs for anomaly detection.

Based on experiments by [8], an anomaly detection system based on log key sequences alone can achieve high accuracies in some datasets, such as the HDFS dataset published by [2]. Similarly, the work by [9] showed that their log key sequence anomaly detection technique can achieve high accuracy on High Performance Computing (HPC) system logs. However, for finer levels of scrutiny, a similar sequence modeling approach can be applied to numerical features that can be extracted from logs, as demonstrated by [3] and [8].

2.3 Concept Drift

An important issue to consider in anomaly detection is concept drift, which is the change in the statistical distribution of a target variable over time. This is especially relevant for semi-supervised anomaly detection systems that are trained on normal data from some baseline period.

An anomaly detection model that is trained in an offline manner on data from a baseline period will see a reduction in accuracy over time. Specifically, when the normal system behavior changes, becoming significantly different from its behavior during the baseline, we’ll commonly see a rise in false-positives,

where normal samples are mistakenly identified as anomalies by the system. Therefore, an effective anomaly detection system needs to be able to adapt to concept drift.

Relatively few previous work on anomaly detection considers concept drift. It is often treated as a separate goal from obtaining a model that learns well and yields high accuracy. Regardless of the model used for detection, the concept drift adaptation always involves updating the model over time.

In a naïve approach, one can simply train a new model from scratch based on a new set of data. However, this approach is not very efficient. Ideally, one would be able to update a model incrementally by retraining on just the new, recent samples, rather than the entire set of baseline data.

Most traditional machine learning techniques are not designed to allow for incremental update. As such, one often has to modify existing algorithms to allow such capability. In [21], a general framework for concept drift adaptation is proposed for clustering based anomaly detection systems, involving the update of baseline exemplars (centroids) with new normal samples with some appropriate weights.

Neural networks on the other hand, are naturally able to learn in an incremental manner. That is, the training data does not have to be considered as a whole. During training, we can divide the training data into small batches, each of which is fed through the model separately to change the system weights a small amount, in a direction that minimizes the loss function. Therefore, to adapt to concept drift, one can simply maintain a small set of new, recent normal samples and retrain the model on that set periodically [8]. There have been previous works that propose more complex mechanisms for adapting to concept drift by adaptively increasing the number of neurons in the network [18]. However, if not managed properly, this approach could lead to uncontrolled growth of the model.

3 - Problem Statement

In this project, our goal is to build an anomaly detection system that can identify anomalies based on analysis of computer-generated logs.

We focus on semi-supervised log anomaly detection, as we believe this setting to be applicable to most real-world systems, where we won't have labeled anomaly data, but could obtain data that is considered normal or sufficiently normal from some baseline period based on the system administrator's evaluation.

Another problem we tackle is concept drift, which is especially important for computer systems which are dynamic and often change over time. In this paper, we evaluate and propose a concrete strategy for effectively updating and retraining our model in an online manner to adapt to concept drift.

4 - Proposed Solution

Following the works of [8][9], we take a sequence modeling approach to log analysis. We treat our goal of learning normal log sequences as a language modeling problem, the goal of which is to learn the structure of a language by building a model that can take in previous tokens (words or characters), and predict what the next token is expected to be based on those previous tokens.

In our application, we seek to build a system that can predict what the next log key is based on a series of previous log keys. For anomaly detection, we train a model on normal sequences from some baseline period. Then, in the detection phase, we apply the model continuously to predict the next expected event. If the new event that arrives agrees with our model's prediction, then it is flagged as normal, otherwise it is treated as an anomaly.

In addition to being applied to sequences of discrete values like log keys, our system can also be applied to continuous, numerical sequences, such as multi-variate time-series of measurements or features. That is, the model predicts the next expected values of those features based on their values from some previous historical window. If the difference (measured by mean-squared-error) between the actual

values and the model's prediction is within some threshold, then it is considered normal. It is abnormal otherwise.

Our anomaly detection system has three components: Data Preprocessing Module, Core Anomaly Detection Model, and Online Update and Retraining Module. We describe each of these in depth in the following sections.

4.1 Data Preprocessing Module

In the data preprocessing phase, we parse the raw logs to extract various features and log keys as described in Section 2, so that they can be processed by our anomaly detection model. Recall that *log key*, *message type*, and *event type* are used by different researchers to refer to the same thing. For the rest of this paper, we will use the term *log key*.

Together, the log keys will form a log key sequence, which represents the underlying execution sequence of the code that printed those log keys. We'll denote the set of all distinct log keys as $K=\{k_1, k_2, k_3,\ldots,k_n\}$, where there are n distinct log keys total.

Our goal is to train a model that can predict the next log key based on a recent sequence of log keys in some historical window of size w (which is a hyperparameter that can be tuned). Therefore, we need to segment our log key sequence into multiple input-output pairs, where each input consists of the log keys in the historical window, and the output consists of the next log key.

To demonstrate this, with a window size w=3 and an observed log key sequence of [k_{14}, k_{21}, k_6, k_{28}, k_4, k_{35}], we will derive the following 3 input-output pairs:

- [k_{14}, k_{21}, k_6] $\rightarrow k_{28}$
- [k_{21}, k_6, k_{28}] $\rightarrow k_4$
- [k_6, k_{28}, k_4] $\rightarrow k_{35}$

Finally, we convert the log keys in both the input and output into one-hot vector representations, such that they can be ingested by the anomaly detection model and be treated as a categorical variable (with number of categories equal to the number of unique log keys).

For numerical sequences, we take a similar approach. From our logs, we derive multivariate time-series of the form $[\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}, ..., \boldsymbol{v_t}, ..., \boldsymbol{v_m}]$, where $\boldsymbol{v_t}$ is a vector (or tuple) storing the values of all the relevant features at time t, and m is the total number of time steps. Then, we segment this in a similar fashion as the log key into multiple input-output pairs.

The feature vectors do not need to be converted to one-hot as they don't take on categorical values. However, different features may take on different numeric ranges. As such, we need to normalize the features such that they have the same range. For each feature, we calculate the mean (μ) and standard deviation (σ) of its value across the baseline period. Then, for each instance x of the feature, we compute its normalized value x' as follows:

$$x' = (x - \mu) / \sigma \qquad (1)$$

This normalization is performed for each value of each feature. This normalization technique is referred to as *z-score normalization*, and is useful when one doesn't know the maximum and minimum possible value of an attribute a priori [11]. This is the scenario we're dealing with, as our training data consists of points from only the baseline period, which most likely will not have data points spanning the entire possible range for each feature.

4.2 Core Anomaly Detection Model

Our system is modular, in that different types of models can be used as the core anomaly detection model, as long as it is compatible with the input and output format from the Data Preprocessing Module described previously.

The most common sequence modeling technique used is the Long Short-Term Memory (LSTM) neural network. LSTM is a type of Recurrent Neural Network (RNN) that uses three multiplicative gates—input gate, output gate, and forget gate—to control gradient flow and alleviate the vanishing/exploding gradient problems encountered when training RNNs on long sequences. Previous works [4][8] have used deep-LSTM for log sequence modeling with promising results. Therefore, we built LSTM-based models as baseline for comparison.

While our LSTM-based models yielded good accuracies, they suffered from long training time. So, we set out to identify an architecture that improves the runtime while yielding equivalent or better accuracies.

The first strategy we undertook to reduce run time is to swap out the LSTMs for Gated Recurrent Units (GRU), which is an alternate type of RNN that has fewer parameters than LSTMs. While this did improve the training time, it led to a drop in accuracy.

Next, following the recent trend in using CNNs for sequence modeling, we built a deep CNN model that uses 1D convolutions. This led to drastic improvements in run time due to the ability of CNNs to be parallelized when performing convolutions with multiple filters. Unfortunately, this model also resulted in the worst accuracy when compared to the RNN based models such as LSTM and GRU.

From these experimental architectures, we gained a few insights. Firstly, CNNs can in fact learn some spatial information based on the ordered events, and can do so extremely efficiently through parallel convolutions. However, its accuracy is sensitive to the choice of filter sizes. LSTMs, on the other hand, is naturally able to capture relationships in sequences and have relatively stable accuracies over different hyperparameter choices as shown by [8]. Lastly, we confirmed the conclusion by [4] and [8] that, when it comes to LSTMs, a deeper model can achieve higher accuracies than a shallower model, but this is at the cost of increased training time.

Based on these observations, we set out to design an architecture that captures both CNNs ability to efficiently extract spatial features in a parallel fashion, and LSTMs superior ability to learn temporal

relationships in sequences. The result is a hybrid CNN-LSTM architecture that uses variable filter sizes and causal convolution to extract features which are then fed into an LSTM to make predictions. This model, which we call "CausalConvLSTM" is shown in Fig. 4.

The CNN portion of the proposed architecture uses filters of different sizes in parallel to extract features across different span of the input sequence, alleviating the concern of not specifying a suitable filter size for the dataset.

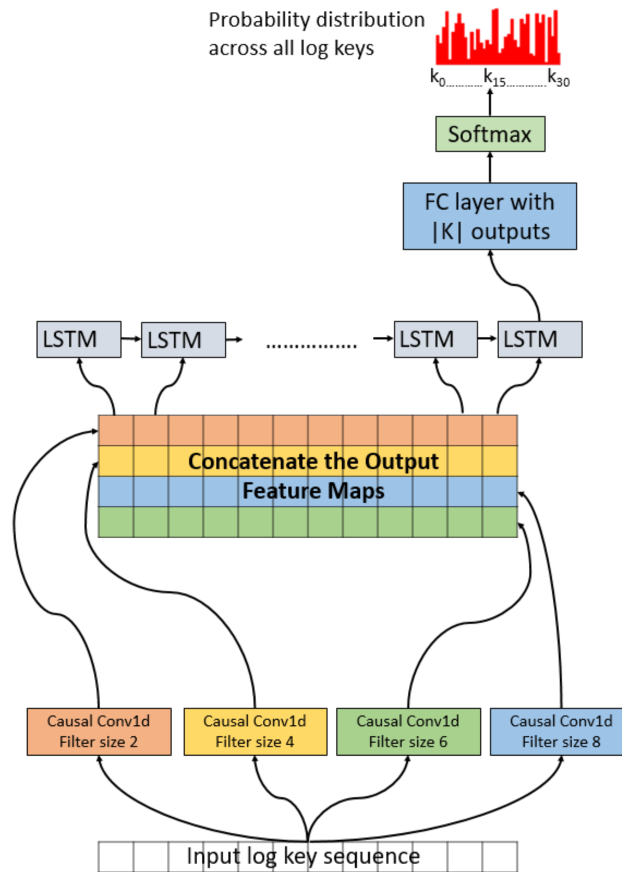

Fig. 4. Proposed CausalConvLSTM Architecture

Furthermore, the use of causal convolution ensures that the implication of ordering is preserved even after we concatenated all the features maps from the output of variable-sized convolutions together. This is illustrated in Fig. 5.

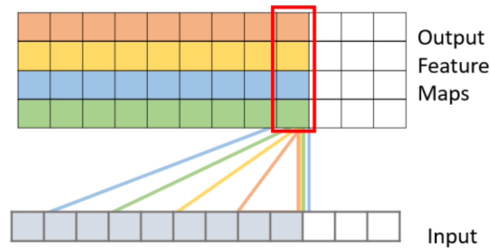

Fig. 5. Causal Convolution as Used in Our Architecture

Since the ordering remains meaningful in this output feature map, we can then use a LSTM to ingest it in order, from the first location to the last. After ingesting the sequence of rich features from the CNN, the LSTM's hidden state vector is passed to a fully connected (FC) layer with same number of nodes as there are log keys. Finally, the output of the FC layer is fed through a softmax function to produce a probability distribution function across all the possible log key values. This probability distribution function is compared with the actual one-hot encoding of the next log key from the training sample, and the categorical cross-entropy loss function is used to determine the error gradient used to update the model weights during training.

Then, in detection stage, we feed our model the most recent log entries within a historical window of size w. The model processes this sequence and outputs a probability distribution function across all possible log keys. We rank the probabilities from most to least likely, and if the new log key that just arrived is among the top g most likely predicted by the model, then it is treated as normal. Otherwise, it is treated as an anomaly, and an alert is generated. Note, g is another hyperparameter that can be tuned, and is essentially similar to a detection threshold. The smaller the g the more selective, and the larger it is, the less selective. A g of 1 would mean we only consider log keys that match the model's single top prediction as normal.

A model similar to the one shown in Fig. 4 can be used for anomaly detection on multi-variate numerical time-series. The only difference is that the FC layer will have number of nodes equal to the number of features, and the softmax activation function will be omitted. The model's output will be its prediction of the next set of values for those numerical features, and will be compared against the actual values using mean-squared-error (MSE) loss function to determine the error gradient used to update the model weights during

training. In detection stage, we feed our model the most recent numerical feature vectors within the historical window, and the model outputs a prediction of what it expects the values of the numerical features to be next. We calculate the mean-squared-error (MSE) between the model's prediction and the actual observed values of those numerical features, and if the MSE exceeds some threshold, then it is deemed anomalous. Otherwise, it is deemed normal. The threshold to use for the MSE is a hyperparameter that can be tuned. We will discuss how to establish suitable values for it in the performance evaluation section.

Based on our experiments, this hybrid model achieves comparable accuracies as the deep LSTM-only models, while requiring significantly less training time.

4.3 Online Update and Retraining Module

In addition to screening new messages and alerting operators when an anomaly is detected, our proposed system allows online training and model update to adapt to changes in system behavior over time (concept drift).

By using a neural network as our core model, we can take advantage of neural networks' natural ability to learn in an incremental manner. That is, rather than taking the entire training dataset as a whole in the learning process, it can update system weights iteratively based on small subsets of the training data. As such, we can retrain the model on a few new samples to make minor updates without having to retrain the model from scratch based on the entire training set with the new samples added. Based on this observation, previous works have proposed adapting to concept drift by incrementally retraining the model on false-positive samples identified by the operator [8]. However, no previous work we are aware of provides concrete detail on how this retraining should be done.

To determine a concrete and robust retraining strategy, we proposed and evaluated many different online retraining strategies. Based on our evaluation, an effective and generally applicable strategy is to trigger retraining when the FP rate calculated from a rolling window of the most recent events exceeds some specified threshold.

5 - Performance Evaluation

As mentioned previously, our system is modular in that different core anomaly detection models can used as long as its input and output format is compatible with those produced by the Data Preprocessing Module. Similarly, the Online Update and Retraining Module can adopt different retraining strategies regardless of the core anomaly detection model used, as long as it is a neural network based model.

To quantify and compare the performance of anomaly detection, we use the popular metrics Precision, Recall, and F-Measure. These measures are defined in terms of the count of True-Positives (TP), False-Negatives (FN), True-Negatives (TN), and False-Positives (FP) as specified by the equations below:

$$Precision = TP / (TP+FP) \quad (2)$$

$$Recall = TP / (TP+FN) \quad (3)$$

$$F\text{-}Measure = 2*Precision*Recall / (Precision+Recall) \quad (4)$$

Note, in the context of anomaly detection, “positive” means anomalous, and “negative” means normal. The precision quantifies what portion of the points identified by the model as anomalous are actually anomalous (per the ground-truth label). The recall quantifies what portion of the points that are actually anomalous the model successfully detected. The F-measure, given as a harmonic mean of precision and recall represents a holistic measure that takes into account both precision and recall [8]. For all three measures, the higher the better.

Another measure of interest we use is the false-positive rate, calculated as FP / (FP + TN). This measure quantifies the portion of normal samples that are mistakenly identified as anomalous by the system. We want to keep this measure low, as an increase in the number of false-positives can overwhelm the operator and cause him/her to distrust and ignore new alerts created by the system.

5.1 Anomaly Detection Evaluation

The dataset we used for benchmarking our core model's effectiveness for log key anomaly detection is the HDFS dataset generated and published by the authors of [2]. This dataset consists of over 11 million log entries generated over a period of 48 hours by a 203-node Hadoop cluster running in AWS [2]. The data set has been used in various previous studies on log anomaly detection [2][3][7][8]. After generating these log messages, the authors of [2] worked with Hadoop domain experts to identify and label blocks that experienced abnormal execution. Using these block ids, we can group the messages into 558,221 normal subsequences and 16,838 abnormal subsequences [8].

Following a semi-supervised anomaly detection approach and the work by [8], we train our model on only the first 4,855 of normal samples (less than 1% of all the normal data points available). Then, we use the rest of the normal and abnormal data points for testing and accuracy evaluation.

Across our experiments, we fix the historical window size (w) at 12, since this is the shortest length observed among the 4,855 normal subsequences. For the hyperparameter g that specifies the top places to compare the new sample against in the model output probability distribution, we used 5, as the experiments by [8] showed that with g=5, 99.8% of the next log key in normal sequences is among the top-g log keys predicted by their model. We can increase g, but the potential gain is marginal, and may result an increase in false-negatives. This g parameter is similar to a detection threshold as we discussed previously.

We first implemented a stacked, Deep-LSTM model based on the description by [8]. This model achieved very promising results but had a long training time. In an attempt to reduce the training time, we experimented with various models including a Deep-GRU, a Deep-CNN, and finally our original CausalConvLSTM model. The model parameters and their performance are summarized below.

Table 1. Model Description and Performance Results on HDFS Dataset

Model	Deep-LSTM	Deep-GRU	Deep-CNN	CausalConvLSTM
Layer 1	LSTM with 75 Units	GRU with 75 Units	Conv1D, 64 filters	Conv1D, 64 filters
Layer 2	LSTM with 75 Units	GRU with 75 Units	Conv1D, 64 filters	LSTM with 96 Units
Layer 3	FC Layer with 30 Nodes	FC Layer with 30 Nodes	Conv1D, 80 filters	FC Layer with 30 Nodes
Layer 4	-	-	Conv1D, 80 filters	-
Layer 5	-	-	FC Layer with 30 Nodes	-
Training Time	28s/epoch	25s/epoch	16s/epoch	15s/epoch
Precision	0.8968	0.8710	0.8949	0.8959
Recall	0.9971	0.9959	0.9619	0.9972
F-Measure	0.9443	0.9293	0.9272	0.9438

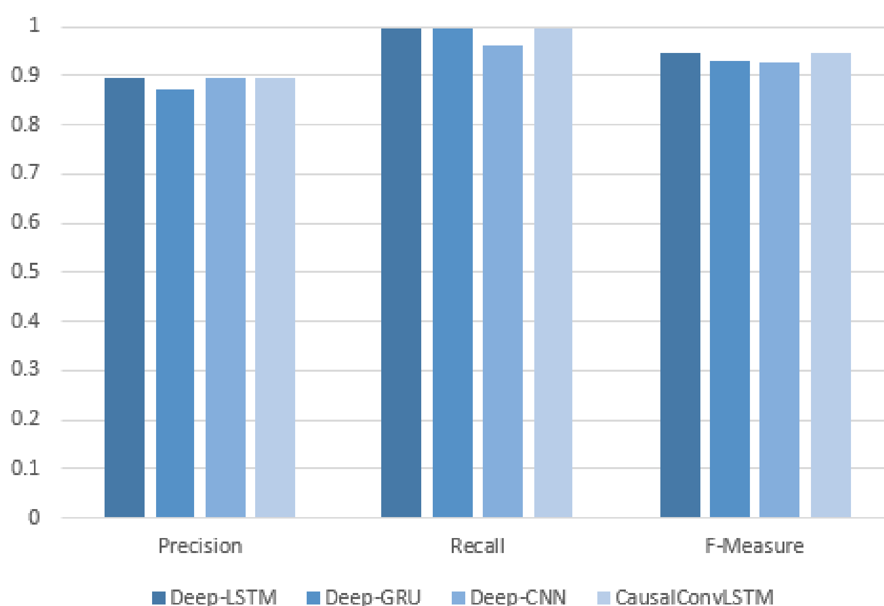

Fig. 6. Model Accuracies on HDFS Dataset

As these results show, the stacked, Deep-LSTM still maintained the best accuracy with a F-Measure of 0.9443. However, the training time of the Deep-LSTM model was also the longest of all the models compared. The Deep-GRU had similar training times with worse accuracies. The Deep-CNN architecture leads to significant reduction in run time due to its ability to take advantage of parallelization, however, this

purely CNN-based model performed the worst of all 4 models compared. Finally, our proposed CausalConvLSTM model achieved results that are essentially equivalent to the Deep-LSTM model, while requiring almost half the training time.

To evaluate the effectiveness of our proposed model when used for numerical sequence modeling, we needed a different dataset. The HDFS dataset has very few numeric features, and was not useful for benchmarking numerical sequence modeling for anomaly detection [8].

The dataset we were able to obtain was the Canadian Institute for Cybersecurity's Intrusion Detection Evaluation Dataset (CICIDS2017), published by the authors of [12]. This dataset was generated from a test network over a 5-day period (Monday-Friday), during which both normal and various attack traffic was simulated. Throughout the simulation, packet-capture (PCAP) logs were collected. The researchers then used their network packet analyzer CICFlowMeter to extract various numerical features from the PCAP logs. The result is a multivariate time series, with the values of multiple features at each timestamp. The data was also labeled to indicate normal traffic as well as the various types of attacks.

For our evaluation, we used the data from the first day (Monday) of the simulation as baseline. This day contained only normal traffic. We used 80% of this day's traffic for training and reserved 20% as a validation set, which is used to help us establish detection thresholds for the mean-squared-error (MSE). The reason is that we want to obtain prediction errors on normal data points the model was not trained on. Therefore, after training the model on the training set, we apply it to each point in the validation set and calculate the MSE between the model prediction and the ground-truth. This gave us a list of model prediction errors across the validation set. Fig. 7 shows a histogram of the MSE values when our CausalConvLSTM model is applied to the validation set.

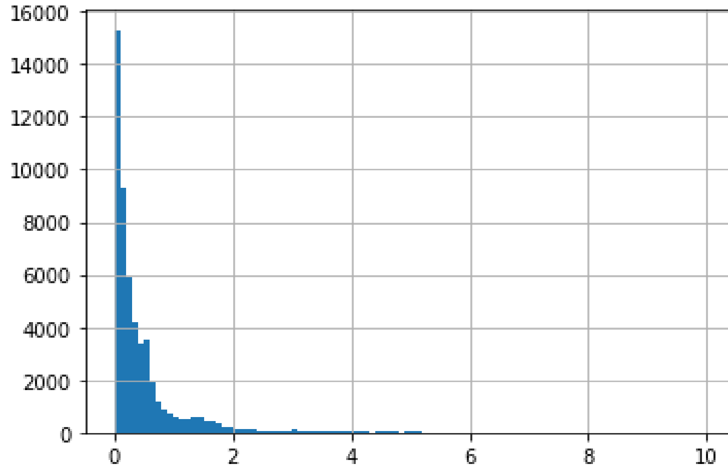

Fig. 7. Histogram of MSEs from Validation Set (CICIDS2017, CausalConvLSTM Model)

Based on Fig. 7, we realized that the MSE losses is highly skewed. As a result, we cannot use means and standard deviations to establish the threshold. Instead, we chose to use the quantile statistic to help us establish the detection threshold. Specifically, we found a threshold set at the 90% quantile of the validation set MSE values gave us the best F-Measure.

For testing, we picked a day from the dataset that contained the most diverse attacks. Specifically, the data from Wednesday, which included normal data mixed with various types of attacks including Slowloris, Slowhttptest, Hulk, GoldenEye, and Heartbleed. We treated all the attacks as anomalies (or "positive") instances in our metric calculation, and the rest as normal (or "negative") instances. Our CausalConvLSTM model achieved an F-Measure of 0.76, which is slightly better than a GRU model we also implemented for comparison, which had an F-Measure of 0.74. However, the CausalConvLSTM model required almost half the training time compared to the GRU model, when trained on the same workstation. The results are summarized in Table 2 and Fig. 8 below.

Table 2. Model Performance Results on CICIDS2017 Dataset

Model	GRU	CausalConvLSTM
Precision	0.77	0.84
Recall	0.72	0.69
F-Measure	0.74	0.76
Training Time	239 s/epoch	151 s/epoch

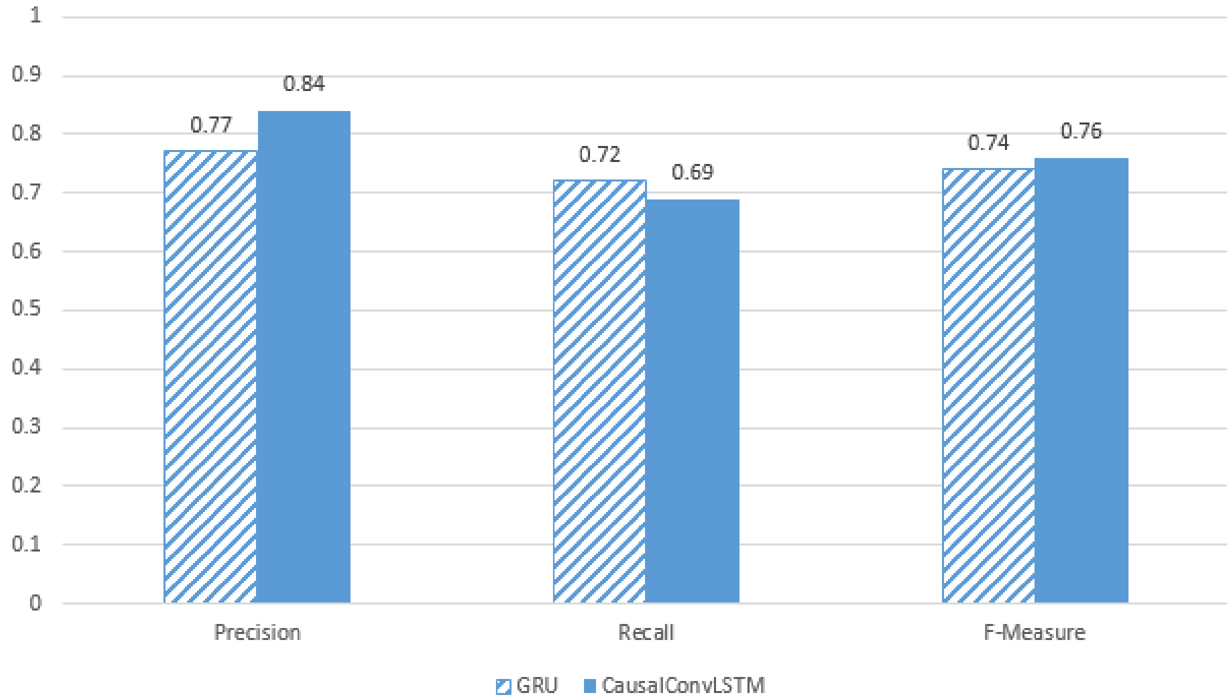

Fig. 8. GRU vs CausalConvLSTM on CICIDS2017 Dataset

Our experiment results show that rather than having a deep homogenous model with multiple layers of the same architecture (either all RNN or all CNN), it is worth considering a hybrid model with mixed architecture. Designed properly, a hybrid model can achieve equivalent or better results while requiring less training time, allowing us to take advantage of the strengths of different architectures.

5.2 Online Update and Retraining Evaluation

For evaluating drift, we needed a dataset that actually exhibits change in normal system behavior over time. Based on our preliminary experiments, the HDFS Dataset proved to be unsuitable for drift analysis, as the normal system behavior is relatively stable over time. This could be due to the fact that the logs from that dataset only spans about 2 days, which is far too short a time period for significant system change to take place. The same is expected of the CICIDS2017 dataset, which spans only 5 days.

For this evaluation, we identified a dataset published by the authors of [10], which consists of system logs generated by a Blue Gene/L supercomputer ("BGL Dataset"). This dataset consists of 215 days' worth of logs, involving over 4 million log messages, of which 7.33% were labeled as abnormal by domain experts. This dataset was used by the authors of [8] for the purpose of drift evaluation.

In a setup similar to that proposed in [8], we assume there's an analyst who is alerted when an anomaly is detected, and who can flag the alert as a false-positive if the entry is actually determined to be normal based on the analyst's evaluation. In our retraining system, we want to save these labeled false-positive entries and use them to update the model later on. These entries represent examples of new normal behavior that our model should know about. As for the true negatives, the model does not need to be retrained on them, because it already knows to treat them as normal. The remaining question is, when should we retrain the model.

We implemented different retraining strategies. We chose as our baseline the first 10% of data, and used the normal data points from the baseline period for training the model. Then, we used the rest of the 90% of data for testing. Fig. 9 below compares the resulting overall FP rate on the entire test set, when different retraining strategies are used.

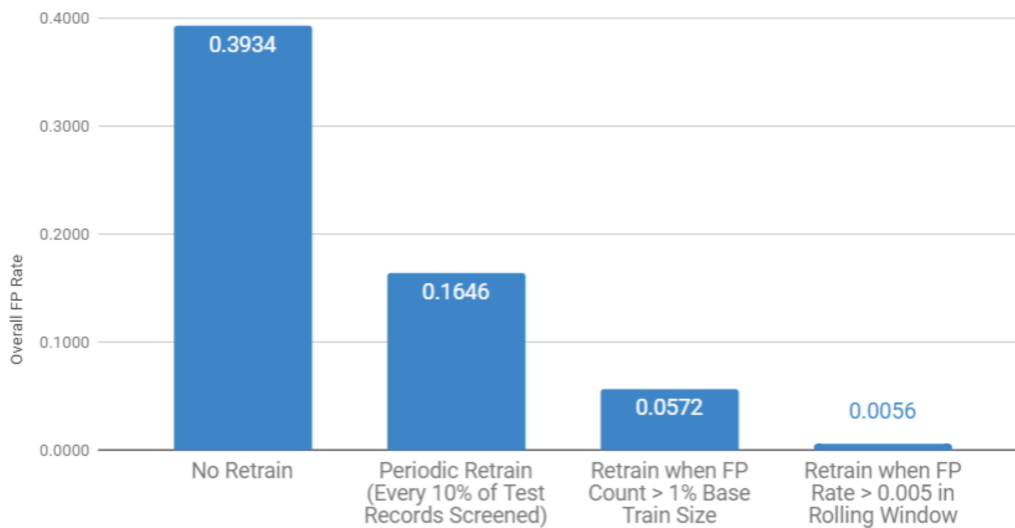

Fig. 9. Overall FP Rate on Entire BGL Test Set with Different Retraining Strategies (GRU AD Model)

The first retraining strategy we tested is to trigger a retrain periodically either based on time elapsed or number of records screened since the last retrain. A downside of this approach is that it has no regard to the FP count or rate, and may therefore retrain even when there's no need to. Additionally, the selection of the retrain interval has significant effect on the performance.

The second approach we experimented with is to have a system that triggers a retrain based on some threshold on the FP count. A retrain is triggered whenever the accumulated FP count since the last retrain exceeds the threshold. We noticed that for this approach, the choice of the FP count threshold again has significant effect on the overall accuracy.

The next approach we experimented with is based on the FP rate. A retrain is triggered whenever the FP rate exceeds some threshold. While experimenting with this scheme we noticed that there are multiple ways to define the FP rate, and how the FP rate is defined has significant impact on the model performance. Three different ways to define the FP rate are as follows: cumulative FP rate, fixed window FP rate, and rolling window FP rate.

The *cumulative FP rate* is calculated by keeping a count of all the false-positives and true-negatives of the system since the beginning of the detection (testing) phase, and calculating the FP rate based on those cumulative counts at each step.

The *fixed window FP rate* is calculated by dividing the time into fixed, non-overlapping increments and tallying the false-positives and true-negatives within each increment to calculate the FP rate.

The *rolling window FP rate* is calculated by tallying the false-positives and true-negatives within a moving/rolling window that encompasses only the most recent events.

Based on our experiments, the cumulative FP rate is not suitable as it misses many momentary spikes in false-positives due to an averaging effect. If there was a prior period of good accuracy followed by a sudden increase in the number of false-positives in quick succession, those new false-positives still would not be sufficient to raise the cumulative FP rate significantly.

FP rate based on fixed, non-overlapping windows are better at capturing momentary spikes in the number of false-positives. However, we're left to the mercy of how the windows are divided, and may or may not capture the highest FP rate if an increase in the number of false-positives happens to land on the boundary between two windows. On the other hand, a rolling window that slides over the timeline would capture all possible divisions ensuring we'll cover a window that has the highest number of false-positives. Based on these observations, we decided that a FP rate based on a rolling window is the most appropriate for use with our retraining strategy. Therefore, we propose calculating the rolling window FP rate at each time step, comparing it to a threshold, and triggering a retrain if the FP rate exceeds the threshold.

From our experiments, a retrain strategy that is triggered when the rolling FP rate exceeds a certain threshold resulted in the lowest overall FP rate of 0.0056 (0.56%). We also observed that the overall FP rate will stay near the threshold we've set. This makes the threshold more meaningful than a threshold on the FP count, as a system administrator is likely to have an idea of what a desirable FP rate might be for his/her system.

Fig. 10 below shows the accuracy measures over time with this retraining strategy. We see that the precision, recall, F-Measure and overall accuracy all stayed relatively high, while the FP rate stayed very low. This indicates that our retraining strategy is effective.

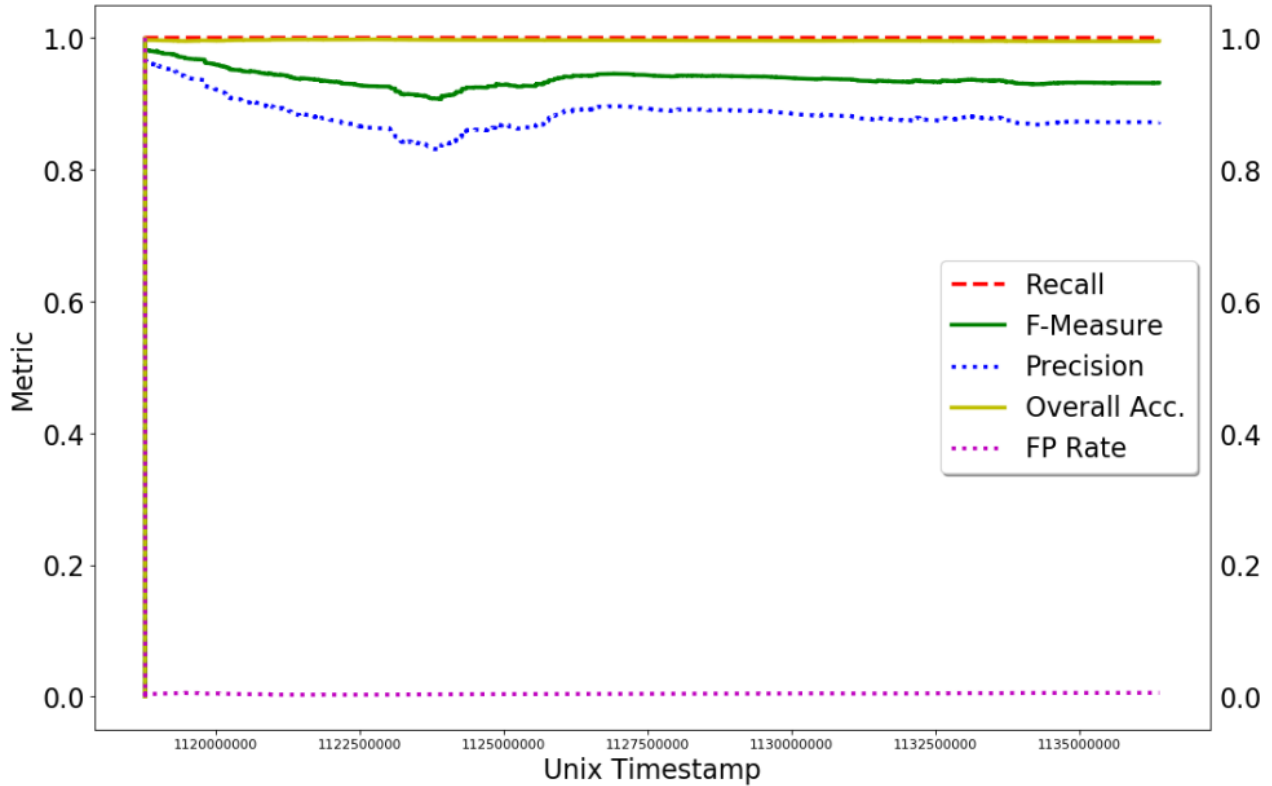

Fig. 10. GRU Model Metrics over Time, with Retraining Based on Rolling Window FP Rate

Lastly, we compared the runtime of the system with and without retraining. Without retraining, the amortized runtime to screen each event is about 1.1 milliseconds, when tested on our mid-range personal workstation. With retraining, the amortized runtime to screen each event is about 0.03 milliseconds longer than for a system without retraining, when tested on the same workstation. This difference in runtime is insignificant considering the improvement in accuracy obtained. Furthermore, as pointed out by [8], the retraining could be done in parallel to detection using a copy of the old model, so that detection time will not be affected at all. The models can then be swapped when retraining is complete.

6 – Discussion and Lessons Learned

In this chapter, we take a step back and review some lessons learned from the various experiments conducted as part of this project. This includes reviewing the other anomaly detection techniques that we experimented with which do not take a sequence modeling based approach.

6.1 Alternate Approaches (Non-Sequence Based)

As discussed in the background section, there are many anomaly detection techniques. However, not all anomaly detection techniques are suitable for our specific problem of anomaly detection from computer logs. Statistical modeling and proximity-based anomaly detection approaches were precluded from our application because they don't perform well in high-dimensional space [14]. On the other hand, clustering based techniques and reconstruction-based techniques have been shown to be effective for anomaly detection from computer logs [2][6][7][19]. As such, we also implemented and evaluated these techniques, applying them to the same datasets we have used to evaluate the sequence modeling based approaches.

Specifically, we evaluated K-Means to represent the clustering-based approach, and a neural network autoencoder to represent the reconstruction-based approach. However, as these techniques don't operate on sequences like our proposed CausalConvLSTM, the data needs to be preprocessed in a different manner. To apply these methods to the HDFS dataset, we follow the approach by [2] and [3] to create an event count vector for each block id, which stores the occurrences of each type of event associated with that that block from the logs. Each event count vector then represents a feature vector. For both the K-Means and Autoencoder based approach, we train the models on 4,855 normal event count vectors parsed from the first 100,000 log entries [8], and test on the event count vectors parsed from the rest of the logs.

For K-Means, we used the same approach discussed in the background section for anomaly detection, as outlined below:

1. Perform K-Means clustering on the training set to derived *k*-clusters and their associated centroids. These clusters and centroids will represent our baseline.
2. For each new test instance *x*, calculate its distance to the closest baseline centroid.
3. If the distance exceeds our threshold *T*, flag the instance as anomalous. If the distance is less than the threshold *T*, the instance is considered normal.

The target number of clusters (*k*) and the threshold (*T*) are hyperparameters. Since we perform clustering on only normal training data, *k* can be considered to be the expected number of types of normal system behaviors. System knowledge or previous, related application can provide some hint as to the appropriate ranges of *k* to use, or one can simply experiment with different *k* values to see which one yields the best results. We consulted previous works [5] and [6] for approximate ranges of *k*, and then simply tried all the values. The results are summarized in Fig. 11. Note, as we have not settled on an appropriate distance threshold, we initially set it as the average distance of each baseline data point to its respective cluster centroid, plus three standard deviations. This is based on the assumption that these distances follow a normal distribution, and 99.7% of the observations are expected to lie within 3 standard-deviations, according the “68-95-99.7” rule.

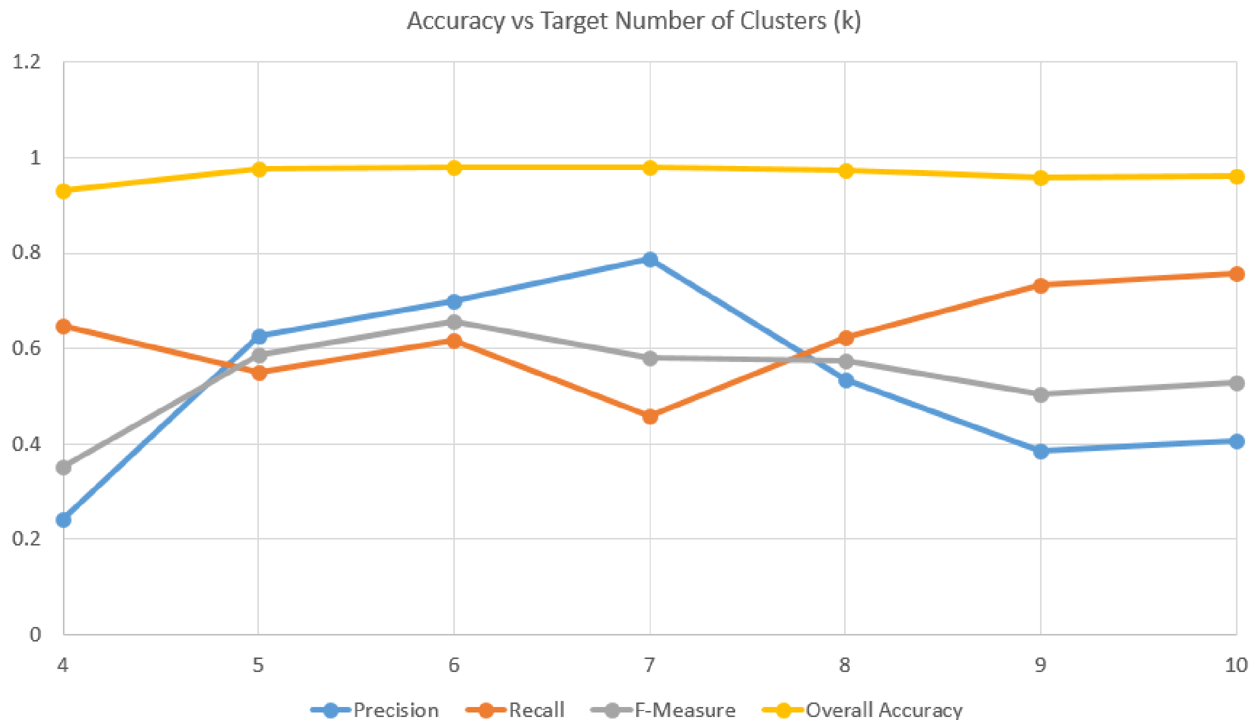

Fig. 11. K-Means Accuracy vs Target Number of Clusters (HDFS Dataset)

We found that *k*=6 gave us the best accuracy based on the overall F-Measure. Therefore, we determined that this is the most suitable selection for this dataset.

The next parameter we need to determine is the threshold (*T*). For setting detection thresholds, one typically rely on confidence intervals of well-known statistical distributions (such as a Gaussian) [8], which was what we did initially. Unfortunately, our experiments showed that the distances of data points from their centroids do not follow a normal distribution; therefore, our initial threshold setting based on the "68-95-99.7" rule was not appropriate. As such, we experimented with different distance thresholds, and found a threshold set at near the 99th percentile of the distances of the validation set data points to their nearest centroids to work the best. With this selection, we obtained a precision of 0.72, recall of 0.61, and F-measure of 0.66.

Next, we created a deep multi-layer perceptron autoencoder (MLP-AE) that is also trained on the event count vectors from the baseline period. As the frequency of different types of events can vary (some

events naturally occur much more often than others), we needed to normalize the counts of each type of events to bring them in the same range. We do so by using z-score normalization [11] as discussed previously. The model is then trained to minimize the reconstruction losses when applied to event count vectors of normal execution sequences. At detection stage, we extract the event count vectors of new sequences and feed them into the autoencoder to get a reconstruction. Then, we calculate the reconstruction loss between the reconstruction and the input. If the reconstruction loss exceeds a certain threshold, it is deemed anomalous. Otherwise, it is deemed normal.

To establish the threshold for the reconstruction loss, we applied the autoencoder to a validation set, which consists of normal data points the model was not trained on. This gave us an idea of the range of thresholds to try. In the end, the threshold that gave the best performance was at around the 99th percentile of the reconstruction losses when the model is applied to the validation set. This gave us a precision of 0.67, recall of 0.66, and F-Measure of 0.66.

The following figure compares the accuracy measures of clustering-based approach (K-Means), reconstruction-based approach (MLP-AE), and sequence modeling approach (CausalConvLSTM) applied to the same HDFS dataset.

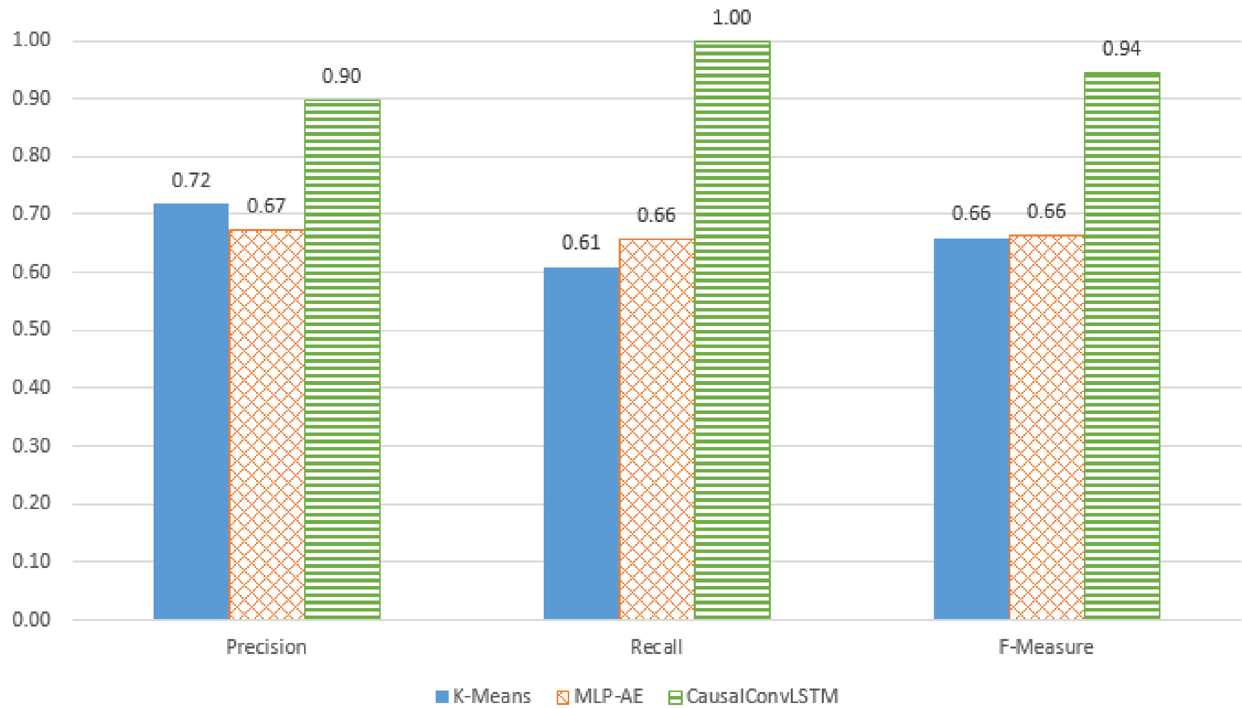

Fig. 12. Clustering, Reconstruction, and Sequence-Based AD Comparison (HDFS Dataset)

Based on this experiment, we observe that the sequence modeling approach far out-performed the clustering and reconstruction based approach. This is somewhat expected because the feature used by our K-Means and deep MLP-AE models are the event count vectors. Since the event count vectors simply tally the number of each type of events that occurred for that execution sequence, the ordering information is not preserved. Therefore, the decision made by the K-Means and deep MLP-AE models do not take ordering in to consideration. The sequence-based approach (CausalConvLSTM) on the other hand, ingests the sequence of event types directly; therefore, it is able to consider the order of event types, and discover sequential relationships. As the ordering of events is important for the execution path anomalies present in this data set [2], it is no surprise that the sequence modeling based CausalConvLSTM performed better.

To further evaluate these different classes of anomaly detection techniques, we also compared them using the CICIDS2017 dataset. As described previously, this dataset contains numerical features

generated by a network packet analyzer [12]. The timestamp in the dataset allows us to order and treat these data points as a multi-variate time-series, as we've done in the sequence modeling approach described previously. Alternatively, we could consider the set of features at each timestamp individually (independent of its surrounding time steps), and try to make a determination based on the information from that point in time alone; this is the approach we'll take when applying the clustering-based model and the reconstruction-based model.

As before, we performed z-score normalization to ensure all the features are in similar ranges. We train the models on the baseline data from the first day (Monday), which consisted of only normal traffic.

For the clustering-based anomaly detection, we run the K-means algorithm on the baseline training data with k=6, grouping the baseline data points into 6 clusters and obtaining the centroids of each of these clusters. These clusters and centroids represent our baseline, normal behavior. Then, in detection stage, we screen new entries by calculating their distance to the closest centroid. If the distance exceeds a certain threshold, it is deemed anomalous. Else, it is deemed normal. Through our experiments, we found a threshold set at around the 97th percentile of the distances between the validation set data points and their nearest baseline centroid yielded the best accuracy.

For the reconstruction-based anomaly detection, we built a deep MLP-AE with input and output layers both with number of nodes equal to the number of features. The bottleneck of the MLP-AE is set at 64% the size of the input. This forces the MLP-AE to learn a reduced-dimension representation of the higher-dimension input, from which it can reconstruct the original input with minimal loss. The MLP-AE was then trained on the normal baseline data. Then, in detection stage, we screen new entries by feeding them through the MLP-AE to obtain a reconstruction. We then calculate the MSE reconstruction error between the reconstruction and the input. If the reconstruction error exceeds a threshold, it is deemed anomalous, else it is deemed normal. Through our experiments, we found a threshold set at around the 97th percentile of the validation set reconstruction losses to give the best accuracy.

We applied our clustering, reconstruction, and sequence-based anomaly detection systems to the CICIDSD17 data from Wednesday, which included normal data mixed with various types of attacks including Slowloris, Slowhttptest, Hulk, GoldenEye, and Heartbleed. The accuracy results are shown in Fig. 13 below for comparison.

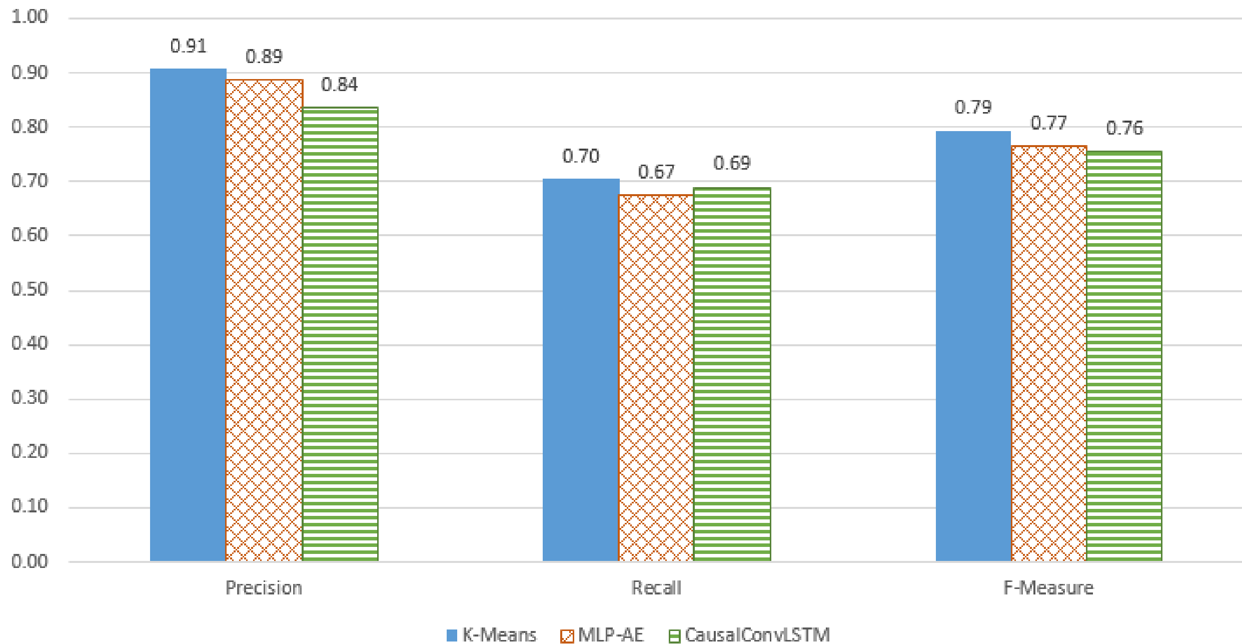

Fig. 13. Clustering, Reconstruction, and Sequence-Based AD Comparison (CICIDS17 Dataset)

Surprisingly, on this dataset, the K-Means based anomaly detection model outperformed deep MLP-AE and CausalConvLSTM. This is in contrast to the HDFS dataset, where the K-Means based approach performed the worst. The fact that our sequence modeling based CausalConvLSTM performed worst here is especially surprising, since it is able to consider sequential relationships, rather than treat each data point as an isolated, independent instance, which was how it was treated by the MLP-AE and K-Means models. Based on some literature review, we came across another paper [20], which also experimented with a sequence-modeling based approach on this dataset. Their experiments also showed that a simpler, frequency-based method out-performed a more complex LSTM-based model [20], and concluded that this is most likely because there isn't much valuable information encoded in the sequences in this dataset. The

data set was created by a packet analyzer, so each data point already summarizes multiple events that took place over a short time window, and this information was sufficient for detecting the types of anomalies involved in this dataset.

While a sequence modeling based approach is more general in that it can capture any sequential relationships that may exist, for the CICIDS17 dataset, the sequence information proved to be unimportant. Therefore, by including information from multiple time steps, we're likely diluting the discriminating power of information that can be gained from a single time step alone. This is likely the reason that a sequence-based approach performed slightly worse than those that do not consider sequence information, as demonstrated by our experiments as well as those by [20].

6.2 Lessons Learned

An important lesson learned from this project is the importance of evaluating the data itself early on. This includes inspecting the raw data as well as attempting to understand the underlying behaviors represented by the data.

The larger a data set, the higher the chance that something could go amiss. For instance, while parsing the BGL data set, we inspected the first few lines of the log to get the general format, and implemented our parser based on the assumption that the rest of the data will follow the same format. Then as extra precaution, we ran our parser on a subset of the log file (the first 1,000 lines) and made sure it performed correctly. Unfortunately, when we proceeded to run the parser on the entire set of over 4 million log entries, we encountered various corrupted log lines, including timestamp fields with unexpected text (spilled over from other fields), unexpected line break character '\n' in the log level field, as well as missing fields. To fix these issues, we needed to include special cases in our parsing logic to handle when things go awry. The CICIDS2017 dataset also contained inconsistencies. Specifically, the CSV file published by the author contained lines with no field values, but simply the comma separators. Also, some labels used a version of the dash symbol that is not recognized by UTF-8 encoding. The

lesson learned is that, when it comes to large volumes of data, what can go wrong will more likely go wrong, so we should anticipate and take the appropriate precautions, such as making our data preprocessing code more robust to handle special cases.

Once we've taken care of these low-level issues with the raw data itself, we also need to understand the behavior of the data. For example, we initially tried to use the HDFS data set for drift analysis, as well as for benchmarking the numerical-sequence modeling effectiveness of our model. However, after initial experiment, we discovered that the HDFS dataset did not actually exhibit drift behavior; that is, the normal system behavior stayed relatively constant throughout the 2-day simulation done by the authors [2]. Additionally, these logs contained only a few numeric parameters, and they occurred very infrequently, resulting in insufficient data to perform meaningful sequence modeling. The original creator of the HDFS dataset [2] also stated that they only identified and labeled the execution anomalies by identifying the sessions (i.e. block_id) that went through abnormal execution. They did not consider any of the numerical features when labeling anomalies. It is for these reasons that [8] only applied log key anomaly detection to the HDFS dataset, and used separate datasets for benchmarking numerical sequence modeling and drift adaptation.

Through our series of experiments, what we discovered is that there is no one-size-fit-all solution for log anomaly detection. That is, not all anomaly detection techniques can be applied to all types of logs with great result. Our sequence modeling based approach takes a step toward creating a more general solution, as it can ingest sequences of events directly rather than relying on the user to engineer the best features that capture the inherent relationships in sequences. However, there's no guarantee that it will always outperform custom solutions hand-picked and designed for specific applications.

7 - Conclusion & Future Work

In this project, we evaluated various techniques used for log anomaly detection. We built upon previous anomaly detection techniques that seek to model log sequences directly using neural networks. Based on observations from our experiments, we designed an original hybrid neural network architecture called “CausalConvLSTM” that takes advantage of CNN’s ability to efficiently extract spatial features in a parallel fashion, and LSTM’s superior ability to learn the relationships in sequential data. Through our experiments, we showed that our model can achieve high accuracy for anomaly detection after training on only a small amount of normal log data. Furthermore, our model requires less training time than previous state-of-the-art architectures, while achieving comparable accuracies. Additionally, we address the concept drift problem by performing evaluations of different online retraining strategies for neural network based anomaly detection systems, and propose a concrete strategy that was effective in maintaining high accuracies and low false-positive rates as demonstrated by our experiments. Future work could involve applying our proposed system to additional types of log data or the incorporation of a module to help users more easily diagnose identified anomalies.

LIST OF REFERENCES

[1] A. Chuvakin, K. Schmidt, and C. Phillips, *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management*, 1st ed. Waltham, MA, USA: Elsevier, 2013.

[2] W. Xu, et al., "Detecting large-scale system problems by mining console logs," Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, New York, NY, Oct. 2009.

[3] S. He, et al., "Experience Report: System Log Analysis for Anomaly Detection," 27th International Symposium of Software Reliability Engineering, Ottawa, ON, Canada, Oct. 2016.

[4] R. Malaiya, et al., "An Empirical Evaluation of Deep Learning for Network Anomaly Detection," International Conference on Computing, Networking and Communications (ICNC), Maui, HI, Mar. 2018.

[5] Lima, M., Zarpelao, B., Sampaio, L., Rodrigues, J., Abrao, T., & Proenca, M. (2010). "Anomaly Detection Using Baseline and K-Means Clustering. International Conference on Software," Telecommunications and Computer Networks (SoftCOM). Split, Dubrovnik, Croatia.

[6] Yin, C., Zhang, S., Wang, J., & Kim, J. (2015). "An Improved K-Means Using in Anomaly Detection," First International Conference on Computational Intelligence Theory, Systems and Applications (CCITSA). Yilan, Taiwan.

[7] Xu, W., Huang, L., et al., "Online System Problem Detection by Mining Patterns of Console Logs," Ninth IEEE International Conference on Data Mining, Miami, FL. Dec. 2009.

[8] M. Du, F. Li, et. al., "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning", Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, Texas, 2017

[9] Haque, A., DeLucia, A., Baseman, E., "Markov Chain Modeling for Anomaly Detection in High Performance Computing System Logs," Proceedings of the Fourth International Workshop on HPC User Support Tools, Denver, CO. Nov. 2017.

[10] Oliner, A. and Stearley, J., "What Supercomputers Say: A Study of Five System Logs," Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK. June 2007.

[11] Jiawei Han, Micheline Kamber, Jian Pei. *Data Mining: Concepts and Techniques*. 3rd ed, Morgan Kaufmann, 2012.

[12] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018

[13] S. Nomm and H. Bahsi, "Unsupervised Anomaly Based Botnet Detection in IoT Networks," IEEE International Conference on Machine Learning and Applications, Orlando, FL, USA, Dec. 2018.

[14] P. Tan, M. Steinbach, A. Karpantne, and V. Kumar, *Introduction to Data Mining*, 2nd ed. Pearson, 2018.

[15] S. Yen and M. Moh, "Intelligent Log Analysis using Machine and Deep Learning," chapter in *Machine Learning and Cognitive Science Applications in Cyber Security*, ed. M. S. Khan, IGI Global, May 2019 pp. 154-189.

[16] I. Goodfellow, Y. Bengio, and A. Courvile, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[17] Y. Meidan, M. Bohadana, et al., "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," IEEE Pervasive Comuting, Oct. 2018.

[18] C. Wang, R. Xu, S. Lee, and C. Lee, "Network Intrusion Detection Using Equality Constrained-Optimization-Based Extreme Learning Machines," Knowledge-Based Systems, 2018.

[19] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log Clustering Based Problem Identification for Online Service Systems," IEEE International Conference on Software Engineering Companion, Austin, TX, USA, Mar. 2017.

[20] B. Radford, B. Richardson, and S. Davis, "Sequence Aggregation Rules for Anomaly Detection in Computer Network Traffic," arXiv:1805.03735v2 [cs.CR], May 2018.

[21] W. Wang, "A General Framework for Adaptive and Online Detection of Web Attacks," Proceedings of the 18th International Conference on World Wide Web, Madrid, Spain, April 2009.