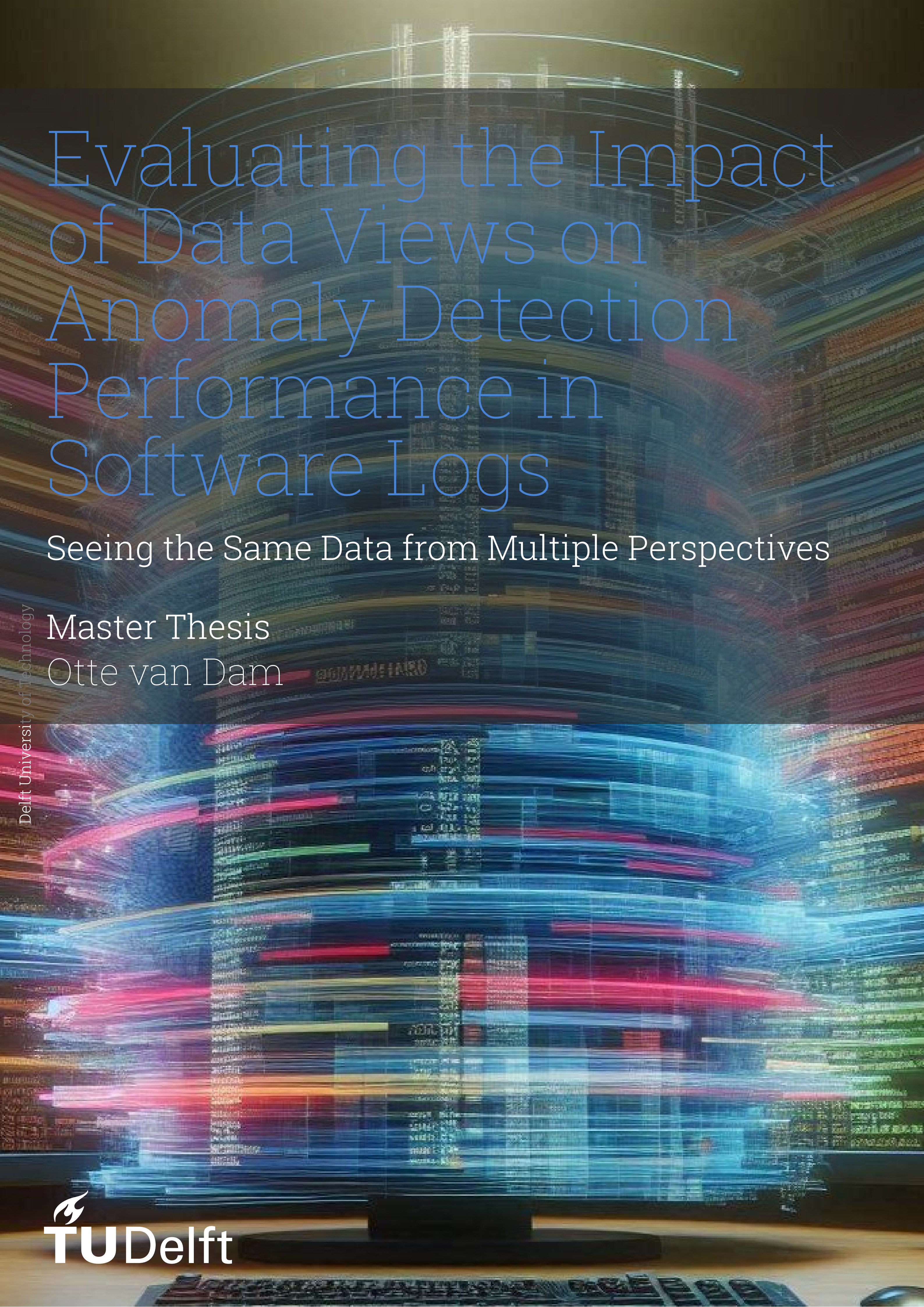


Evaluating the Impact of Data Views on Anomaly Detection Performance in Software Logs

Seeing the Same Data from Multiple Perspectives

Master Thesis

Otte van Dam

Evaluating the Impact of Data Views on Anomaly Detection Performance in Software Logs

Seeing the Same Data from Multiple Perspectives

by

Otte van Dam

Student Name	Student Number
Otte van Dam	5096790

Supervisor:	S. Verwer
Project Duration:	November, 2023 - June, 2024
Faculty:	Faculty Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft

Preface

During the completion of my master's thesis, many people have helped me successfully finish my master's degree in Data Science. These people deserve to be mentioned and thanked.

First of all, I would like to thank my supervisor, Sicco Verwer. I have learned a lot about data science from you, and I always felt that I was given a lot of freedom to investigate what I found interesting. Your thoughts and feedback during our weekly meetings were very helpful and often fun as well. These weekly meetings would not have been such a joy without Hielke, Mahira, and Mirijam. You guys always brought a good mood and interesting discussions, whether about computer science or not. A special thanks to Mahira, who used the same data as I did and was always there when I had a question or just wanted to vent about how annoying EVTX logs are.

My master's thesis marks the end of my studies here in Delft. During my studies, I often had to work in groups. Once you find the right teammates, it all becomes very easy. For my bachelor thesis, I would like to thank Tristan and Louis. You guys made most of the second half of my bachelor's very easy. It is great when your teammates always deliver good work and are always available to discuss the projects we worked on. It is even greater when your teammates become your best friends along the way.

For my master's, I quickly found two teammates who helped me with all the group work. Colin and Suhaib, I would like to thank you for all the hard work you delivered and the fun we had in the meantime or outside of study. It was a great relief once I realized that I would enjoy working with two hardworking teammates again.

Throughout most of my bachelor's and my entire master's, I worked on projects led by Marcus Specht, whom I also want to thank. You provided me with interesting work, the experience from which helped me get a job for after my studies. I learned a lot during my work and always felt appreciated for what I did.

Last but not least, I want to thank my family, who have always been there for me during my time as a student. First, I would like to thank my parents. You always supported me and gave me a loving home that I could also use as my study place. Alongside my parents, my girlfriend Laura has always supported me and has done so for over 10 years. You may not have always understood what the ”weird symbols” on my screen meant, but you always knew what I needed from you.

Otte van Dam
Delft, June 2024

Acknowledgements

To improve the writing of the research, ChatGPT3.5 was used to improve self-written parts of the thesis. No AI was used to generate any research or results, only to improve grammar and spelling.

Abstract

As our world has become increasingly digital and the number of tasks performed by software has grown, so too has the volume of software logs and the importance of cybersecurity. Anomaly detection on software logs is crucial for securing systems and identifying the causes of past attacks. Extensive research has focused on developing effective log parsers and anomaly detection methods. However, the process between obtaining parsed logs and feeding them to the anomaly detection methods has remained unchanged for years. Typically, logs are ordered by their generation time and split into fixed-length timeframes, which are then analyzed for anomalies. This approach can group unrelated logs together while splitting related logs across different timeframes, potentially missing critical context for anomaly detection.

This research explores the concept of views, which are different perspectives used on input data. For example, logs can be grouped by the computer they are generated on, allowing the comparison of computers instead of arbitrary timeframes. The study demonstrates that the performance of anomaly detection methods can vary significantly depending on the views used.

To validate this, two different anomaly detection methods were applied to two different datasets, showing that the effect is not specific to any particular dataset or detection method.

The research discusses three methods for creating these views and proposes a method to suggest the most useful view by using views that contain a lot of different log keys and log values. Additionally, partial views, such as grouping logs by user, are examined. These views, although missing some logs, prove valuable for detecting anomalous behaviour as grouping all logs of a user together and comparing that to other users, helped find the anomalous user within a dataset that contain more than 1400 users in total.

The study also investigates the cause of anomalous behavior in Windows event logs by analyzing the scores of individual logs to identify words present in anomalous logs but absent in normal ones. While this approach has some false positives, it effectively highlighted the timing and consequences of an attack.

Finally, the research explores various ways to combine scores from different views and their effects. Combining all views of EVTX data provides a reliable middle ground, useful when the most effective view is unknown. It was found that taking the highest score from all views results in many false positives and should be avoided. However, using the highest x scores from all views, as long as x is greater than one, does not significantly differ from taking the average score.

Contents

Preface i

Acknowledgements ii

Abstract iii

1 Introduction 1

2 Background 3
- 2.1 Log Parsing . . . 3
- 2.2 Anomaly Detection in Software Logs . . . 4
- 2.3 N-Grams . . . 5
- 2.4 FlexFringe . . . 5
- 2.5 PCA . . . 7
- 2.6 ROC Curves . . . 9
- 2.7 Log datasets . . . 9

3 Literature Review 11
- 3.1 Identifying key ingredients in data pre-processing for machine learning in cyber-security . . . 11
- 3.2 Experience Report: System Log Analysis for Anomaly Detection . . . 12
- 3.3 SFAD: Toward effective anomaly detection based on session feature similarity 12
- 3.4 LogSpy: System Log Anomaly Detection for Distributed Systems . . . 13
- 3.5 Mining Invariants from Console Logs for System Problem Detection . . . 13

4 Research Questions 15
- 4.1 Research Gap . . . 15
- 4.2 **RQ1:** What is the effect of using different views on anomaly detection in software logs? . . . 15
- 4.3 **RQ2:** How can useful views be suggested? . . . 16
- 4.4 **RQ3:** How can different views be combined into a single anomaly score? . . 16

5 Methodology 17
- 5.1 Log Parsing . . . 17
- 5.2 Finding Views . . . 18
- 5.3 Combining Views . . . 20
- 5.4 N-grams Per Log Variant . . . 21
- 5.5 FlexFringe . . . 24
- 5.6 Finding Interesting Words . . . 25
- 5.7 Finding Anomalous Users . . . 25
- 5.8 Evaluation . . . 26
  - 5.8.1 Datasets . . . 26
  - 5.8.2 Timed Windows . . . 27
  - 5.8.3 Manual Inspection . . . 28
  - 5.8.4 Baseline Models . . . 29

6 Results 30
- 6.1 Research Question 1 . . . 30
- 6.2 Research Question 2 . . . 36
  - 6.2.1 Interesting Words . . . 39
  - 6.2.2 Interesting Users . . . 41
- 6.3 Research Question 3 . . . 43

7 Conclusion **45**

8 Limitations **47**

9 Future Work **49**

References **50**

1 Introduction

In the vast landscape of modern software systems, the enormous amount of log data contains a wealth of information that can be used to find anomalies in these systems. The amount of data is however to big for humans to dig through to find these irregularities. Various anomaly detection methods have been made to detect the anomalies automatically.

Many existing methods process data sequentially, dividing it into timed windows that are transformed into event-count matrices. These matrices serve as inputs for supervised anomaly detection methods, which learn from labeled training data to predict anomalies within subsequent timed windows of sequential data. Despite extensive research focus on refining these anomaly detection methods, less attention has been directed towards optimizing how data is presented to these techniques. This aspect deserves exploration as enhancing data presentation could potentially elevate existing anomaly detection capabilities with minimal additional effort.

Current research often encounters limitations in practical industry applications due to the scarcity of labeled data, a consequence of the labor-intensive process involved in annotating millions of logs. Furthermore, access to real-world system data, particularly data reflecting actual attacks, is not freely available. In such contexts, unsupervised anomaly detection methods offer promise by eliminating the dependence on labeled data for predictions. Additionally, there is a lot of value in not only identifying anomalous time windows but also annotating individual anomalous logs. This approach enables analysts to pinpoint the specific causes of anomalies and finding what is happening in the system.

This research aims to advance anomaly detection in software logs by introducing the concept of 'views'. Views represent different perspectives for analyzing data, which can enhance the contextual understanding which is crucial for anomaly detection. For instance, instead of the conventional sequential ordering of logs, grouping data by system components provides a more coherent context. This approach ensures that logs pertaining to similar processes or systems are analyzed together, enhancing anomaly detection accuracy. Views can also include different perspectives like focusing on users or specific systems. This recognizes that users and systems have unique behavior patterns that might not fit neatly into traditional timed windows.

The research only uses unsupervised anomaly detection methods, focusing on predicting anomalies for each individual log to be as practical as possible for industry use. The main focus is on testing how different views affect anomaly detection accuracy and demonstrating their usefulness in real-world situations.

Rather than offering a complete anomaly detection solution, this thesis explores ways to improve anomaly detection systems by presenting data in new and innovative ways. By studying the impact of these 'views' on anomaly detection, the research aims to advance the development of anomaly detection methods in software systems.

2 Background

This chapter will explain the background knowledge necessary to understand the rest of the research.

First, log parsing is discussed, which is a preprocessing step in the anomaly detection pipeline. Next, common anomaly detection methods in software logs are explained to provide an overview of current research methodologies. Following this, n-grams are introduced as one of the anomaly detection methods used in this research. Subsequently, the basics of FlexFringe are covered. FlexFringe is a tool used in this research to test current state-of-the-art techniques. Next, PCA (Principal Component Analysis) is explained, as it serves as one of the baseline methods in this research. Afterwards, ROC curves (Receiver Operating Characteristic curves) are described, as these are used to visualize the results of the anomaly detection methods. Lastly, the log datasets frequently used in anomaly detection research are highlighted as they are also used for this research.

2.1. Log Parsing

To perform anomaly detection on software logs, the logs are typically divided into log keys and log values. This is often done to use log keys to represent a log. Log keys represent the constant part of a log message that remains unchanged across similar logs, while log values are the variable components that differ within these logs. For instance, consider the following log messages:

"Job {Job-ID} has started running on {computer name}".

In this example, "Job Job-ID has started running on computer name." is the log key, while Job-ID and computer name are the log values.

To identify log keys and values within a dataset, the data is parsed using a log parser. Log parsing is a complete research field on its own, but DRAIN [3] is a commonly used parser [2], [4], [7], [15], [18]. DRAIN is often chosen for its high accuracy and fast parsing capabilities.

DRAIN utilizes regular expressions, informed by domain knowledge, to preprocess raw log text. For example, it can detect and isolate the date and time of a log entry, placing this information in a separate column rather than including it in the log keys and values. After preprocessing, DRAIN attempts to find a suitable log group within a tree structure by following specifically designed rules encoded in the tree nodes, such as the length of the log message and the first word within the message. To manage the size of the tree, it has a fixed depth, which limits the number of nodes a log can visit. An example of a parse tree with a depth of 3 can be found in figure 2.1.

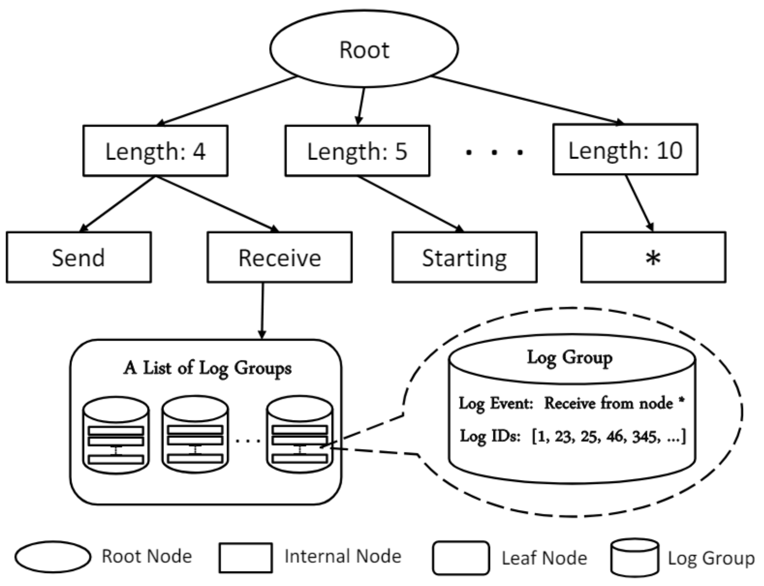


Figure 2.1: Drain Parse Tree with a Depth of 3 taken from [3]

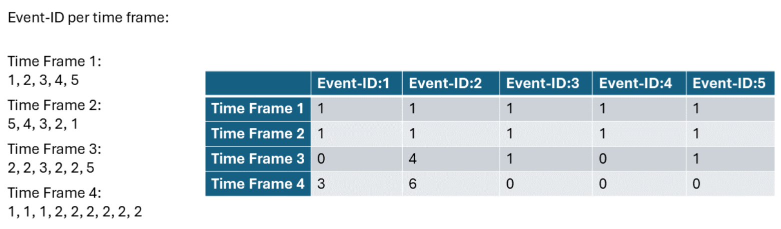
In the leaf nodes of the tree, there are lists of log groups. Each log is assigned to the log group whose log template (the predicted log key) has the highest similarity to the log entry. If the similarity falls below a defined threshold, a new log group is created for that log. The fixed tree depth ensures a significantly faster runtime compared to other parsers, with improvements of up to 81% in runtime efficiency [3].

2.2. Anomaly Detection in Software Logs

Software logs can be challenging for anomaly detection because they often consist of raw text, while most anomaly detection methods rely on numerical values [13], [4], [6]. The sheer volume of text makes text-based vectorization methods like TF-IDF impractical. Consequently, many anomaly detection methods use vectorization based on the event-ID of the log [4]. An event-ID is assigned to the log key of a group of similar logs. For example, all logs following the template ”Job Job-ID has started running on computer name” might share the event-ID 1046.

Event-IDs are ordered by the date and time of the logs and grouped into time windows. For instance, all event-IDs within a half-hour time frame are grouped together. A time frame is considered anomalous if any log within it is identified as an anomaly.

These time frames are then used to create an event count matrix. Each time frame corresponds to a row in the matrix, and each column representing an event-ID. The value at row i and column j indicates the number of times event-ID j appears in time frame i. An example of an event count matrix can be found in figure 2.2



Event-ID per time frame:

Time Frame 1:
1, 2, 3, 4, 5

Time Frame 2:
5, 4, 3, 2, 1

Time Frame 3:
2, 2, 3, 2, 2, 5

Time Frame 4:
1, 1, 1, 2, 2, 2, 2, 2, 2

	Event-ID:1	Event-ID:2	Event-ID:3	Event-ID:4	Event-ID:5
Time Frame 1	1	1	1	1	1
Time Frame 2	1	1	1	1	1
Time Frame 3	0	4	1	0	1
Time Frame 4	3	6	0	0	0

Figure 2.2: Example of an Event Count Matrix

This event count matrix serves as the input for anomaly detection methods, such as decision trees or PCA. These methods return an anomaly score or a classification (anomaly or not) for the entire time frame, but do not provide individual scores for each log entry.

2.3. N-Grams

An n-gram is a sequence consisting of n items from a given sample of data. These items can be characters, words, bytes, or other units, depending on the specific application. For instance the sentence, ”The bunny jumps over the fence” can be split into 4 n-grams of size 3: (The, bunny, jumps), (bunny, jumps, over), (jumps, over, the) and (over, the, fence).

N-grams are traditionally associated with natural language processing (NLP), but they have significant applications in other domains, including anomaly detection [12]. They are particularly useful in anomaly detection because of their ability to capture sequential patterns and their versatility in handling non-numeric data. For instance, in analyzing software logs, categorical values like log keys can be utilized. Take a trigram (a sequence of three items) such as (event-ID 105, event-ID 455, event-ID 295). Such a sequence could signify actions such as data reception, data sanitization, and data utilization within a program’s operations. If such a pattern is common but later an instance is observed where the program receives and uses data without sanitizing it, this deviation from the usual pattern flags as an anomaly.

They are particularly useful due to their ability to capture sequential patterns and dependencies within the data. When data deviates from these usual patterns, it can be identified as an anomaly. N-grams are simple and computationally efficient, and their versatility allows them to be applied to various types of data, making them particularly handy for dealing with non-numerical data.

2.4. FlexFringe

FlexFringe is one of the state-of-the-art techniques that can be used to do anomaly detection. It is an open-source software tool designed to learn finite state automata from traces using a state-of-the-art evidence-driven state-merging algorithm at its core [14]. It takes traces, such as sequences of event-IDs from computer log files, to construct a state machine that models the behavior of that computer. What distinguishes FlexFringe is its algorithm for merging states during the learning process, which is performed heuristically.

Before merging can be performed, the input data is presented as a large tree-shaped automaton, which is called a prefix tree. An example can be seen in 2.3.

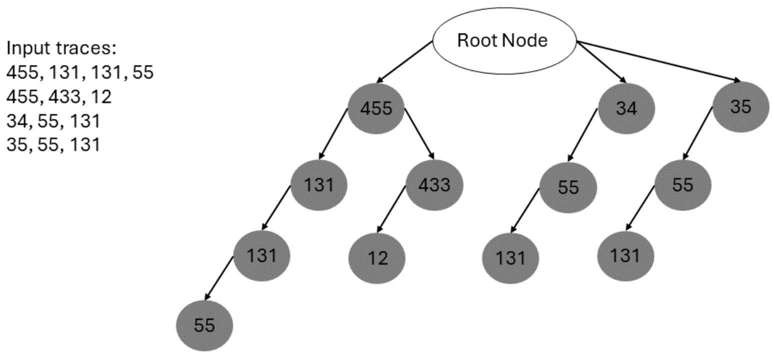


Figure 2.3: An example of a prefix tree.

Each branch in this tree does not merge at any point with another branch. When this tree is constructed, the merging starts.

The state-merging process checks all pairs of states and begins by checking whether one of the states is a sink. Sinks are states with user conditions that are ignored by the merging algorithm. For instance, states that are visited less than 50 times. If neither state is a sink, the algorithm evaluates the consistency of the merge and assigns it a score. The merge with the highest score is executed. If no merge exceeds a set threshold, the model is extended. The completed model of a credit-card transaction can be found in figure 2.4 which is taken from [16].

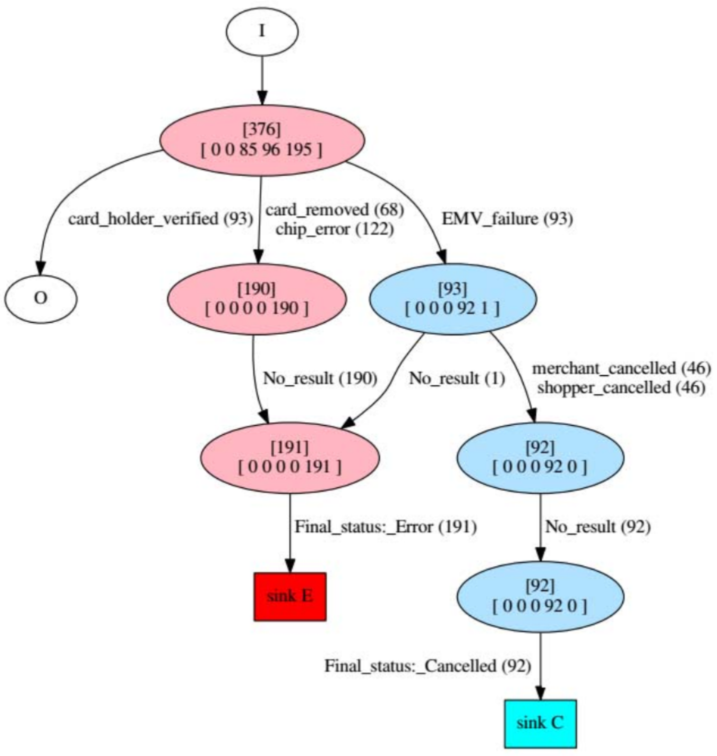


Figure 2.4: FlexFringe model for a credit-card transaction taken from [16].

The figure for instance shows that the model has learned that the outcome of the merchant cancelling or the shopper canceling does not matter for the rest of the process. As stated in [16]: ”Each node holds the total number of traces through the node on the first line, and the number of respectively unknown, approved, declined, cancelled and error traces on the second line.”. This shows that the model learned a deeper understanding of the process that it tries to model.

Once the model is complete, it can predict the state sequence for a given trace. FlexFringe predicts the probability of an input trace by calculating the probability of its state sequence. For each input symbol, it calculates the probability of that symbol occurring in the respective state at each position in the trace. For example, given the input trace (a, b, a, a), it first calculates the probability of symbol ’a’ at the starting state, then the probability of symbol ’b’ in the next state, and so on.

2.5. PCA

Principal Component Analysis (PCA) was first introduces by [5]. It performs linear dimensionality reduction on data using statistical techniques. Data that has 200 attributes can be transformed to a chosen number of so called principle components. This is done while trying to keep as much of the existing variance between the datapoints as possible. This can be useful for many situations. It can for instance reduce the noise in the data or allow data with many dimensions to be reduced to 2 or 3 dimensions so it can be displayed in graphs. For instance, figure 2.5 shows PCA performed on a dataset that contains images of hand written digits.

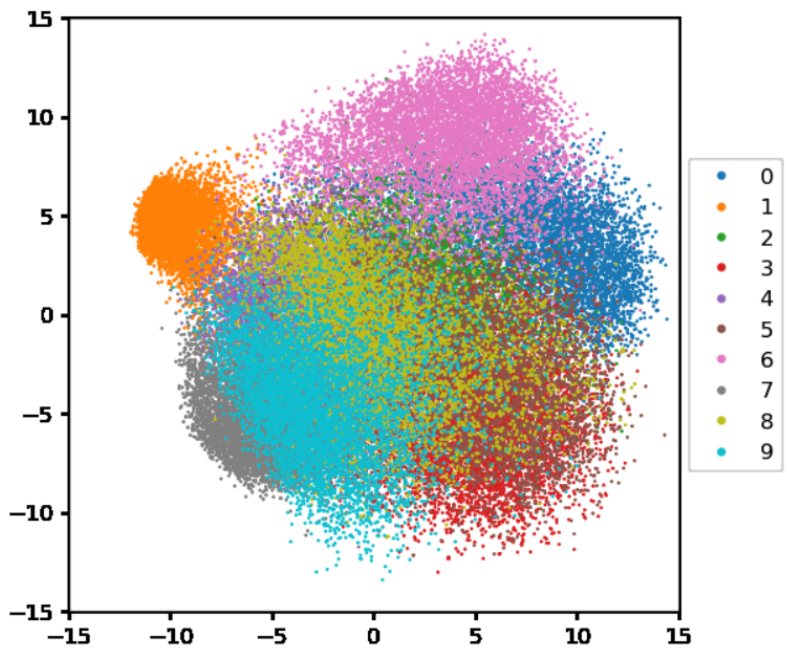


Figure 2.5: PCA performed on handwritten digits. Taken from [10]

It has become however also become a very popular technique in anomaly detection. By first reducing the dimensions while keeping most of the variance in the data and then trying to reconstruct the datapoints with the old attributes, you can find datapoints that are very different from the original data [4]. These datapoints are deviating from the usual pattern and can therefor be seen as anomalies.

A simple example can be the following: There are 5 datapoints that roughly follow the formula [x, x] for any given x, and one datapoint that does not. The datapoints are the following:

	Value 1	Value 2
Datapoint 1	1	1
Datapoint 2	2	2
Datapoint 3	5	6
Datapoint 4	4	3
Datapoint 5	10	11
Datapoint 6	1	11

Table 2.1: Event Count Matrix

Using PCA the data is projected onto a 1-dimensional space. In this case it is roughly on the y=x line. The difference between this projection and the actual datapoint is called the reconstruction error. The projection and the reconstruction error can be seen in figure 2.6.

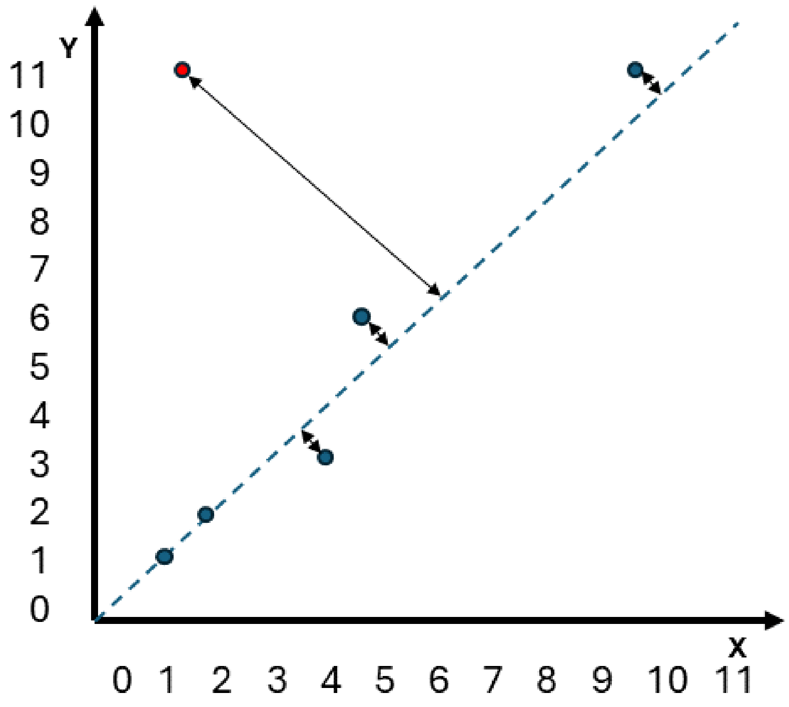


Figure 2.6: PCA Reconstruction Error Example.

The bigger the error, the more the datapoint deviates the more anomalous it is seen as. The calculated construction errors for this example are: [0.01465383, 0.02491413, 0.02111924, 0.12553209, 0.10841881, 1.22720951]. It is clear that the datapoint 6 is the most anomalous.

The datapoints used in anomaly detection on software logs is a row in the event-count matrix [4] [17] where each time frame corresponds to a row in the matrix, and each column represents an event-ID. The value at row i and column j indicates the number of times event-ID j appears in time frame i. An example of an event count matrix can be found in 2.2.

2.6. ROC Curves

Receiver Operating Characteristics (ROC) curves are graphical representations that illustrate the performance of a binary classifier across different thresholds. In the context of anomaly detection, these thresholds serve as cut-off scores: data points are considered anomalous if their anomaly score exceeds the threshold. An example of an ROC curve for a neural network performing anomaly detection on credit card transactions can be found in Figure 2.7.

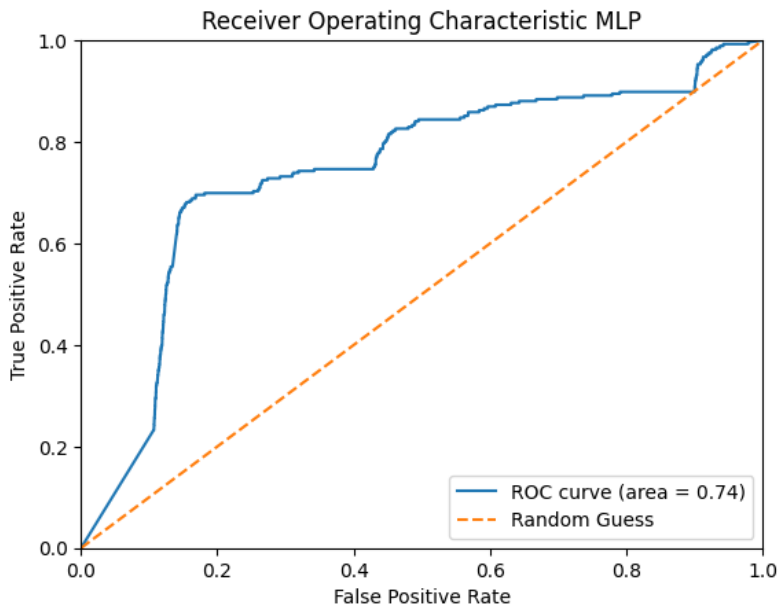


Figure 2.7: ROC curve for anomaly detection on credit-card data using a neural network.

On the y-axis, the true positive rate (TPR) is shown, representing the number of correctly classified anomalies divided by the total number of anomalies. On the x-axis, the false positive rate (FPR) is shown, representing the number of non-anomalies incorrectly classified as anomalies divided by the total number of non-anomalies. Each threshold yields a different TPR and FPR, which correspond to a single data point on the graph.

The ROC curve illustrates how well the detection method identifies anomalies while minimizing the classification of non-anomalies as anomalies. It can also help determine an appropriate threshold for the classifier. To compare the results of different classifiers, the Area Under the Curve (AUC) can be used. As the name suggests, the AUC calculates the area under the ROC curve. Classifiers that achieve high TPR combined with low FPR obtain higher AUC values than those that do not.

2.7. Log datasets

There are numerous datasets available that consist of software logs, with LogPai being the most widely known platform to contain these datasets. LogPai is an open-source platform designed to facilitate the analysis and processing of log data. It provides a comprehensive suite of tools and methodologies to handle various aspects of log management, from parsing to anomaly detection. LogPai contains many datasets widely used in research to compare log parsers and anomaly detection methods for software logs. Most log datasets are unlabeled due to the difficulty of labeling millions of logs. However, LogPai does include some labeled datasets, such as the BGL and HDFS datasets. The section of LogPai that contains the log datasets is called LogHub [19].

The HDFS dataset [17] consists of system logs generated by the Hadoop Distributed computing network. It contains a total of 11,175,629 logs divided over 575,062 blocks, labeled as normal or anomaly. Blocks labeled as anomalies contain runtime performance problems confirmed by Hadoop developers, totaling 16,838 anomalous blocks. Blocks that are anomalous can be very well detected by 'missing' logs [17]. These logs could for instance be a log that is closing an action. Anomalous blocks can also contain logs that are very rare [17]. Most anomaly detection methods tested on the HDFS dataset use the labeled data to train the model on blocks and then predict which blocks in the test data are anomalous.

The BGL dataset is discussed in [11]. This dataset consists of system logs from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768GB of memory. The dataset contains a total of 4,747,963 logs labeled as benign or with a type of anomaly. Each log entry includes a timestamp, date, node, type, component, and log content.

The primary goal in anomaly detection on software logs, is to identify anomalous areas rather than correctly classifying every single log. Therefore, most detection methods divide the data into windows of a defined length, such as all logs within a one-hour time frame. Models are trained on a training set, and predictions are made for the test set. Windows are predicted to be either anomalous or not, with a window considered anomalous if it contains at least one anomalous log.

3 Literature Review

This section will summarize some of the most important research that was used as inspiration for this research or to get a better understanding of what was currently being done in the research field. Each section represents a paper with in the section only the information that was used as inspiration or as background knowledge.

3.1. Identifying key ingredients in data pre-processing for machine learning in cyber-security

This paper discusses the use of pre-processing of network logs before they are used for anomaly detection. The paper describes 3 distinct methods of pre-processing.

The first is by creating different channels on which anomaly detection can be done. The paper describes 4 used for their research:

1. Overall network as time series.
2. Packets send from a single source.
3. Packets received by a single source.
4. Packets within a single connection.

A subset of the logs represent these channels and are fed to the various machine learning methods to do anomaly detection.

The second method of pre-processing is by characterizing each instance. A server used for video streaming behaves differently than a private user playing chess online. This is done by looking at the flows from a single host or connection and presenting them as the mean and standard deviation of the time series it forms.

The last method of pre-processing is extracting sequential information. Every flow is looked at through its context. This is done by grouping flows in sliding windows of size 5. The last flow in the window is used to determine the label of the window. If the last flow is considered an anomaly, using the context of the other flows in the window, the whole window is seen as an anomaly and vice versa.

The results showed that combining the different channels provided the best results, compared to using only one channel at the time. The channels were combined by combining the sliding windows for every channel into a single row in a dataset for each log.

3.2. Experience Report: System Log Analysis for Anomaly Detection

This paper compares 6 log based anomaly detection methods of which 3 are supervised and 3 are unsupervised. The following supervised methods were compared:

1. Logistic Regression
2. Decision tree
3. Support Vector Machine (SVM)

The following unsupervised methods were compared:

1. Log Clustering
2. PCA
3. Invariants Mining

The methods were compared on 2 different datasets. The HDFS dataset, with 11,175,629 logs and 16,838 anomalies, and the BGL dataset, with 4,747,963 logs and 348,460 anomalies.

The logs of the dataset were parsed and grouped in sequences by three different windows: fixed windows, sliding windows and session windows. Session windows are grouped by logs with the same session identifier. For the HDFS dataset, this is the block_id. The BGL dataset did not have any identifiers and therefor session windows were not used on the dataset.

The log sequences were then used to create an event count matrix. The rows of the matrix represent a log sequence and every column stands for an event, in this case a log key. The value of row i and column j is the amount of times log key j occurs in log sequence x. This event count matrix is fed to the anomaly detection methods to detect anomalies in the log sequences.

After comparing the results of the datasets on the anomaly detection methods, the authors noted the following findings:

1. Supervised anomaly detection methods achieve high precision, while the recall varies over different datasets and window settings.
2. Anomaly detection with sliding windows can achieve higher accuracy than that of fixed windows.
3. Unsupervised methods generally achieve inferior performance against supervised methods. But invariants mining manifests as a promising method with stable, high performance.
4. The settings of window size and step size have different effects on supervised methods and unsupervised methods.
5. Most anomaly detection methods scale linearly with log size, but the methods of Log Clustering and Invariants Mining need further optimizations for speedup.

3.3. SFAD: Toward effective anomaly detection based on session feature similarity

The paper tries to find abnormal web-users based on the sessions of the user. Where a session is a sequences of pages visited during an access. The paper uses PageRank, an algorithm used by Google's search engine to rank web-pages, to determine weight information of web-pages. The pagerank score is calculated as follows:

$$PR(p) = (1-d) + d\sum_{i=1}^{n} \frac{PR(T_i)}{C(T_i)} \tag{3.1}$$

where:

$PR(p)$ is the PageRank of page p,
T_i represents other webpages that point to webpage p,
d is the probability that the user randomly arrives at a webpage,
$C(T_i)$ is the number of links from webpage p.

The pagerank weigths are then used by SimHash, a locality sensitive hashing algorithm, to create signatures for the users. The similarity between users is then calculated using the signatures produced by SimHash. Using the similarity between the users, the users are clustered using fuzzy clustering λ-truncating algorithm. The algorithm uses a similarity matrix and random λ-values to cluster the users. Users that are often clustered alone are seen as abnormal.

3.4. LogSpy: System Log Anomaly Detection for Distributed Systems

LogSpy, an anomaly detection method for distributed system, is introduced in this paper. The method uses a convoluted Neural Network to detect the anomalies. First, the log templates are extracted from the logs. Using the templates, a feature vector is created based on the words in the templates. These features are created using skip-gram and AGNES. The feature are then passed to a CNN to do anomaly detection.

The data is split into different windows. This done by creating optimization windows. These windows are the same as sliding windows but logs that are within range r of the sliding windows and that are connected to the selected log are added to the window. This prevents crucial data from not being considered because they are outside of the window.

3.5. Mining Invariants from Console Logs for System Problem Detection

This paper introduces an innovative approach to anomaly detection through unstructured log analysis, employing invariants mining as a key technique. Invariants mining focuses on identifying linear patterns within system workflows. When a log group deviates from these established characteristics, it is identified as an anomaly. Detection of such anomalies indicates a departure from the expected behavior, highlighting potential issues within the system. By pinpointing the specific invariant constraint that is violated, the method aids in the identification and resolution of failures or faults within the system.

The initial stage of the process involves parsing log files. These files are divided into the log key (referred to as the message signature in the paper) and log values (referred to as parameters in the paper). Log parsing itself is not the focus of the research; therefore, the authors adopted the log parsing method outlined in [1]. The log parser extracts tuples for each log, resulting in a structure that resembles the following:

$$[T(m), K(m), PV(m,1), PV(m,2), ..., PV(m, PN(m)] \tag{3.2}$$

Where:

$T(m)$: is the timestamp of message m
$K(m)$: is the log key of message m
$PV(m, x)$: is the x^{th} log value of message m
$PN(m)$: is the number of log values in message m

The log abstraction helps to extract the log key and valueswhere these values represent specific parameters. These parameters are defined by the combination of the log key and value index. Considering a log key like: *"New job added to schedule, jobid=[], priority=[]"* with value index of 0, the parameter is the job-ID and the parameter value might be 3856. The job-ID could be present in multiple logs within the dataset. Logs sharing the same job-ID often relate to the system's execution flow, even if they have different log keys, and the job-ID may be located at varying value indices. The paper introduces a method that automatically determines whether two parameters correspond to the same program variable. When two parameters are identified as representing the same program variable, they are referred to as "cogenetic". Two parameters are considered cogenetic based on the following observations:

1. If parameters P_a and P_b have the same range of values, $V_r(P_a) = V_r(P_a)$, or the value range of one parameter is a subset of of the other, $V_r(P_a) \subseteq V_r(P_b)$, then they are cogenetic.
2. If parameters P_a and P_b have a large joint set $V_r(P_a) \bigcap V_r(P_b)$ there is a high probability they are cogenetic.
3. The larger the length of each parameter value in the joint set is, the higher the probability that the parameters are cogenetic. Where the parameter value length is the number of characters of the value.

Once cogenetic parameters are identified, log collections are formed, each containing logs that share the same parameters, such as a collection of logs containing a specific job-ID. Within these collections, distinct groups are created, each containing logs with identical parameter values for the cogenetic parameters, like logs sharing the same job-ID

For each log collection, a matrix is generated, where each row represents the log group associated with a specified parameter value, such as the log group for job-ID 3467. Each column corresponds to a unique log key found in the log collection across all log groups. The value of X_{ij} denotes the number of occurrences of log key j within log group i. This matrix serves as the basis for anomaly detection through invariants mining.

The identification of the invariant space and row space of the matrix is achieved through singular value decomposition and analysis. Within the invariant space, emphasis is placed on uncovering sparse invariants, as these are more comprehensible for system operators. Compact invariants are avoided, as they do not directly contribute to meaningful workflow structures.

When new logs are received, they undergo parsing and grouping using the same methodology as the training data. A log key count vector is created for each group, similar to the vectors used in the matrix. This vector is then cross-referenced with the corresponding invariants. Any violation of an invariant by the vector is flagged as an anomaly, signaling potential irregularities in the system.

4 Research Questions

This section will discuss the research questions that will be addressed in this study. First, the research gap is explained by outlining the current research methodology and its limitations. Following this, the three research questions are presented, each accompanied by a detailed explanation.

4.1. Research Gap

Almost all anomaly detection methods applied to software logs organize their data by time and date, then divide the data into windows based on a time frame. Predictions are made using a context of sequential logs without considering whether this context is relevant. For instance, if there are many logs from different users, it might be more effective to group logs from the same user together and use those logs as context, as they are more likely to be related to each other than logs from various users.

This research will focus on the impact of using different ways of viewing the data and how to identify these views. A way of viewing the data will be referred to as a ”view.” For example, you could have a user view and a sequential view. To explore this, the following research questions will be addressed:

4.2. RQ1: What is the effect of using different views on anomaly detection in software logs?

This research question examines the impact of utilizing different views on the same data and anomaly detection method. It seeks to determine whether considering how data is ordered and presented to the detection method can enhance the effectiveness of anomaly detection. This is done to find out if views are supposed to be considered when performing anomaly detection. The expectation is that the view of the input data has a big influence on the outcome.

The effect will be measured by giving anomaly detection methods the same data with 2 different views. First the sequential data is given to the anomaly detection method and then a view that is grouped on a component or a block. The difference in results is then used to determine what the effect of the different views are on the performance of the detection methods. To ensure that this is not only the case for a single dataset or detection method, it is done on 2 datasets and 2 detection methods.

4.3. RQ2: How can useful views be suggested?

This research question delves into various methods for identifying useful views and aims to determine their utility without executing each view and resorting to score calculations based on labeled data. It also explores the potential of anomaly detection on partial data to yield valuable insights. The primary objective is to consistently identify potentially useful views. As the number of views increases, prioritizing the most impactful one becomes increasingly challenging, we will therefor try to find a guideline that finds the most useful views.

Finding views will be done by implementing log grouping which has been highlighted in previous research but still misses the implementation details. Examples of using columns are also shown as well as an example of using domain knowledge to find views. Next to that, the use of partial views will be studied by trying to determine an anomalous user by comparing users to each other. It is expected that there are different ways of finding (partial) views and that we might be able to determine which views could be useful by looking at the amount of different logs they impact and the difference in the logs that are within these views. It is also expected that we can find interesting users by comparing them against each other.

4.4. RQ3: How can different views be combined into a single anomaly score?

This research question investigates methods for integrating different views into a single anomaly score and examines the effects of combining these views and effects of combining the views in different ways.

Views will be combined by taking the average of multiple views or by taking the average of the highest x scores from different views on the same log. The effect will be examined by comparing the result of the combined view with the scores of other views. It is expected that the combined view will always perform medium, while the highest x scores might find a value for x that will detect anomalies that are not detected by most views but can be found using only the highest few scores.

5 Methodology

This chapter will discuss the methodology used for this study. First the log parsing used is discussed. Then the 3 different methods of finding views are explained and shown. Next, methods of combining views are discussed. Afterwards, the first anomaly detection method, n-grams per log, are explained in detail. Subsequently, the methodology used for FlexFringe is explained. Then the technique of finding interesting words within the logs of the dataset. The following section provides an explanation of how anomalous users can be found within the dataset. Lastly, the methods of evaluating the research is explained.

5.1. Log Parsing

The datasets utilized in this study consist of unstructured log files. Extracting the log key and values requires a parsing process, which is facilitated by a log parser known as DRAIN. The method is a fixed **d**epth t**r**ee b**a**sed onl**i**ne log parsi**n**g method. It is designed to be a faster alternative compared to other log parsers. While the intricacies of log parsing do not constitute a focal point in this research, interested readers can find specific details on the implementation in [3].

DRAIN is used as the parser in this research because it is the most widely used parser in the research field of anomaly detection on software logs. It is faster than other available options and delivers better or comparable results [3].

For parsing, default values of a similarity threshold of 0.5 and a depth of 4 were applied. The log format utilized depended on the dataset used, for the BGL dataset the following format was used: *<Label>* *<Num1>* *<Date>* *<Component1>* *<Time>* *<Component2>* *<Type>* *<Component>* *<Content>*. In this format, the ”Content” field contains the content with the level at the forefront. The level is the severity of the log message, such as ”info” or ”warning”, and is often excluded from the log message. The level can provide vital information and typically offers limited variations, allowing it to be integrated into the log message and create different log keys for logs with the same message but with a different level. Parsing the arbitrarily picked log:

”- 1119569230 2005.06.23 R22-M0-NB-C:J17-U01 2005-06-23-16.27.10.301258 R22-M0-NB-C:J17-U01 RAS KERNEL INFO 6182400 double-hummer alignment exceptions ”

Results in the following structured log:

Label	-

Num1	1119569230
Date	2005.06.23
Component1	R22-M0-NB-C:J17-U01
Time	2005-06-23-16.27.10.301258
Component2	R22-M0-NB-C:J17-U01
Type	RAS
Component	KERNEL
Content	INFO 6182400 double-hummer alignment exceptions
EventId	5af65199
EventTemplate	INFO <*> double-hummer alignment exceptions
ParameterList	['6182400']

Parsing the evtx data from the APTA and EYE datasets can be really hard. The general log messages are not stored in the logs themselves but windows looks them up if they are viewed within the event viewer. This makes it very hard to extract log keys from the data. Instead, we used the event-id that the evtx data already has pre-defined. To create more views we took a view parameters that are always in each evtx log: proces-ID, thread-ID, activity-ID, security-ID, computer name, source and channel. As these are in every log, they can be used to make views that involve every log in the dataset.

5.2. Finding Views

Certain datasets already contain fields in their logs, such as a process-id in evtx files, which can serve as a criterion for grouping the logs. However, not all datasets feature such columns. To address this, we adopted a modified log grouping approach inspired by the methodology outlined in [8]. This approach facilitates the identification of groupings even in datasets lacking explicit grouping criteria.

Within each log, parameters are represented by combinations of the log key and their respective index values. For example, a log key might be structured as *"Job <Job-ID> was started at <Date and Time>"* with parameters indexed at 1 and 2. Some parameters are shared among logs; for instance, if a subsequent log key reads *"Job <Job-ID> was ended after <execution time> seconds"* the parameter at index 1 likely corresponds to the same job ID parameter in the preceding log key. When the values of two parameters are very similar, they are called co-genetic. When the values of two parameters closely resemble each other, they are referred to as 'co-genetic'. Logs can be grouped based on the shared values of parameters present across multiple log keys. For instance, logs can be grouped on the Job-ID present in the log.

Determining whether parameters are co-genetic depends on the range of values they include. Parameters are considered co-genetic if they meet these conditions:

1. The value ranges of the parameters are either the same, overlap significantly, or one is a subset of the other.
2. Both sets of values contain at least 10 different values.
3. Each value consists of at least 3 characters.

The first rule ensures that the parameters likely refer to the same kind of value, like a job ID. The second rule reduces the possibility of mistakenly identifying parameters as co-genetic because one has a very narrow range and the other has a much broader range, potentially causing the narrower range to be a subset of the broader one. The last rule decreases the likelihood of parameters with distinct meanings sharing similar values due to those values being only a few characters long. For instance, if job-ids are named somewhere between 0 and 100 and computers are as well, then these values might seem cogenetic even though they are about completely different items.

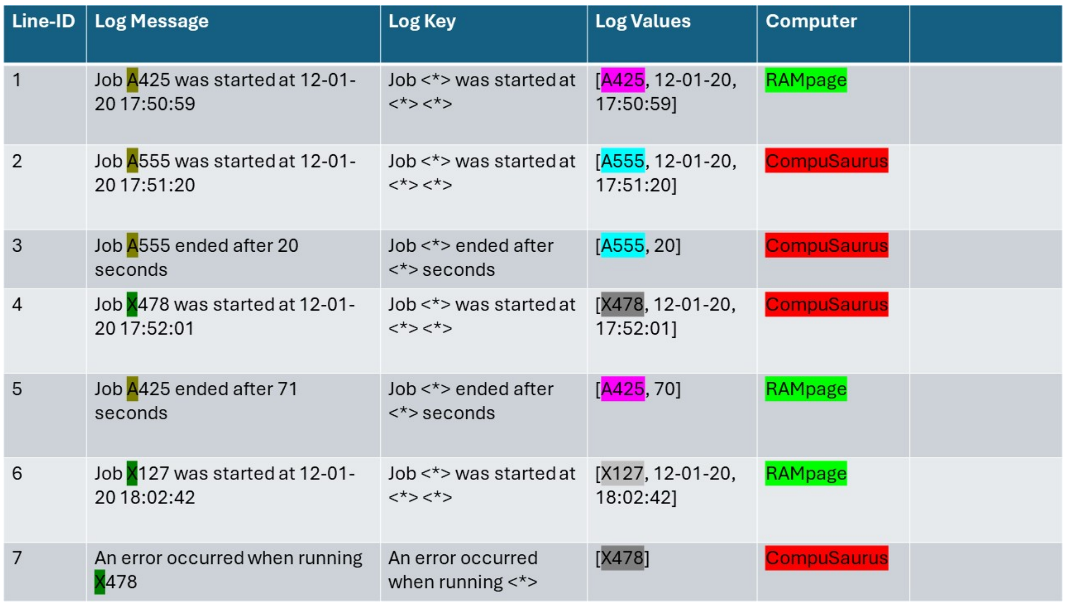
Once the cogenetic parameters are found, logs can be grouped according to the values of the parameters to create a view. For example, a set of co-genetic parameters could represent job IDs, and all entries containing one of these job IDs would be utilized to form a subset. The subset could hold more log keys than those explicitly associated with the co-genetic parameters.This broader inclusion ensures the presence of logs that might be considered anomalies on their own.

For instance, if an error log follows a particular job where something went wrong, it might not be categorized within any co-genetic parameter group due to the size of the value range of the parameters being to low. However, it’s crucial to include such logs when examining a specific job ID, as they indicate issues during that particular job. Subsequently, the subset of logs is grouped on the values of the co-genetic parameter group.

In a dataset, there may be numerous co-genetic parameters, and creating a view for each one could be excessively time-consuming. Therefore, analysts need to prioritize and decide which views are worth exploring further and are likely to yield valuable insights. To assist analysts find interesting views, they are sorted by the amount of unique log keys that are part of the parameter group. A smaller amount of log keys lowers the possible combinations which lessens the chance of anomalous combinations.

Another approach to creating views is by leveraging domain knowledge. For instance, if an analyst suspects the presence of anomalous users within the dataset, they can employ regular expressions (regex) to identify these users. For instance, if users are always present in the logs after ’username:’, going through the logs and finding all instances of that regular expression finds all the logs that are a part of a users behaviour. Logs of these users can then be grouped together to create a new view.

An example of how different views can be made can be found in figure 5.1.



Line-ID	Log Message	Log Key	Log Values	Computer	
1	Job A425 was started at 12-01-20 17:50:59	Job <*> was started at <*> <*>	[A425, 12-01-20, 17:50:59]	RAMpage	
2	Job A555 was started at 12-01-20 17:51:20	Job <*> was started at <*> <*>	[A555, 12-01-20, 17:51:20]	CompuSaurus	
3	Job A555 ended after 20 seconds	Job <*> ended after <*> seconds	[A555, 20]	CompuSaurus	
4	Job X478 was started at 12-01-20 17:52:01	Job <*> was started at <*> <*>	[X478, 12-01-20, 17:52:01]	CompuSaurus	
5	Job A425 ended after 71 seconds	Job <*> ended after <*> seconds	[A425, 70]	RAMpage	
6	Job X127 was started at 12-01-20 18:02:42	Job <*> was started at <*> <*>	[X127, 12-01-20, 18:02:42]	RAMpage	
7	An error occurred when running X478	An error occurred when running <*>	[X478]	CompuSaurus	

Figure 5.1: A set of logs that can be grouped in different ways.

The figure shows logs in sequential order. These logs can be viewed from three different perspectives ,other than the sequential order, with the methods explained in this section:

1. **Computer column in the data:** This perspective uses the computer that generated the log. Since the data has a separate column for the computer, it is easy to create this view.
2. **Job-ID found using log grouping:** Log grouping might find that the same job-IDs are used in multiple log keys. The values in the first index of each log key seem to take on the same values, indicating that the first index of the log keys may be co-genetic.
3. **Job-ID type found using domain knowledge:** This perspective uses domain knowledge. Some job-IDs start with an ”A” while others start with an ”X.” Analysts might know that the letters represent different types of jobs. These different kinds of jobs can be identified using regex.

The grouping of the logs are the following for each view: Computer view:

1. RAMpage: (1, 5, 6)
2. CompuSaurus: (2, 3, 4, 7)

Job-ID view:

1. A425: (1, 5)
2. A555: (2, 3)
3. X478: (4, 7)
4. X127: (6)

Job-ID type:

1. A: (1, 2, 3, 5)
2. X: (4, 6, 7)

The usefulness of each view is dependent on where a failure might happen. For instance, if there is a computer that opens up to much tasks and can not handle the load, the computer view will provide the most insight. However, in this case there is an error happening with a task at log 7. There is to little data to look at only the tasks that start with an X so comparing the jobs to each other might bring the most insight. If there was however a lot of data for both X jobs and A jobs, and it is known t

5.3. Combining Views

Different views can provide various perspectives on the data, and combining these perspectives might offer a more complete picture. Before combining the results from different views, the results are first normalized. This ensures that each view is equally important and prevents any single view from disproportionately influencing the final score due to a high average anomaly score or large deviations in values.

To normalize the anomaly scores, z-score normalization is used. Z-score normalization ensures that the mean anomaly score of each view is 0 and the standard deviation is 1. This is achieved by subtracting the mean value from each score and then dividing each result by the standard deviation.

One way to combine the views is by averaging all the scores from the different views that have a calculated score for the log. This method creates a balanced scoring system: if most views do not detect any anomalous behavior and one does, the average score will remain relatively low. This approach reduces the chance of false positives since multiple views need to have a high anomaly score for a log to be considered anomalous. However, it can also miss anomalous behavior detected by only one or a few views.

Another way to combine the views is by taking the highest score from all the views. If any view detects something anomalous, the log is flagged as anomalous. This approach reduces the chances of false negatives but may lead to many false positives. It also does not differentiate between a log with an anomaly score of 1 across all views and a log with an anomaly score of 1 in just one view. In that case, logs that are anomalous for all logs are labeled as just as anomalous as a log where for all but one view, it is seen as a non-anomaly.

A potential middle ground between using the average score and the highest score can be found by considering the top few highest scores from all views. For instance, one could take the top 3 highest scores for each log. This approach can help identify anomalies detected by only one or two views without generating many false positives, as there might always be a view that assigns a higher score to a log. This method also gives more priority to a log that has high scores across multiple views compared to a log with a high score in only one view.

To test this approach, the combined scores can be calculated by considering the top 1 highest score, the top 2 highest scores, and so on, up to the nth highest score, where n is the total number of views (which equates to the average score). This way, the effectiveness of combining different numbers of top scores can be evaluated to determine the optimal balance between detecting anomalies and minimizing false positives if there is one.

5.4. N-grams Per Log Variant

In this research, one of the techniques employed for anomaly detection is the utilization of n-grams. N-grams use sequences of log keys to identify anomalies within the dataset. One notable advantage of n-grams is their speed in processing large datasets and their use of the context of the logs. For this research it is important that the contexts of logs are used to see what the effect is of using different views. The n-grams are utilized to generate anomaly scores for each individual log entry. This enables analysts to promptly identify potentially anomalous logs, streamlining the process compared to techniques that merely highlight anomalous data sections, leaving the analyst to discern which part of those sections is actually anomalous.

To assign scores to each log, we implement a sliding window mechanism with a size of n and a stride of 1, traversing the data. Within this window, an n-gram is constructed using the log keys of the n logs it includes. For each log we store how often the n-gram has been seen in the data before this log.

An anomaly score is then calculated for each log based on the n-grams it is involved in. First, the total count of all the n-grams, until the point of the n-gram as explained in the paragraph above, is summed up and divided by the number of n-grams to obtain the average count of all the n-grams associated with the log. Since a higher count indicates a more common occurrence, implying less anomalous, we take the inverse of this average count by dividing 1 by it. This results in the final anomaly score for the log.

The calculation of the anomaly score can be formally represented as follows:

1. Average count of n-grams:

$$\bar{C}(L) = \frac{\sum_{N_k \in N(L)} C_L(N_k)}{|N(L)|}$$

2. Anomaly score:

$$S(L) = \frac{1}{\bar{C}(L)} = \frac{|N(L)|}{\sum_{N_k \in N(L)} C_L(N_k)}$$

Where:

- L is the log.
- $N(L)$ is the set of n-grams that log L is part of.
- $C_L(N_k)$ is the count of the n-gram N_k seen before the log L.
- $\bar{C}(L)$ is the average count of the n-grams associated with log L.
- $S(L)$ is the anomaly score for log L.
- $|N(L)|$ is the number of n-grams associated with log L.

When creating a view, for instance using the components in the BGL framework, the data is grouped per component while maintaining the sequential order within each group. The sliding window uses logs exclusively within each component group to prevent overlap between adjacent components. However, a drawback of this approach is that the first and last log of a component is only part of 1 n-gram, making it challenging to determine whether a high anomaly score is attributed to those logs or to other logs within the n-gram. Similarly, the second log and second-to-last log are part of only two n-grams, and so forth. Consequently, some logs have less contextual data available, which may affect the accuracy of anomaly score predictions.

An example could be a component that has the following event-ids in sequential order:

Log	Event-ID
1	455
2	455
3	455
4	455
5	3300
6	599

When using a sliding window of size 3, we get the following n-grams:

N-gram	Occurrences
(455, 455, 455)	4088
(455, 455, 455)	4089
(455, 455, 3300)	2300
(455, 3300, 599)	1

Table 5.2: N-grams of Component and their Occurrences

The first n-gram has been seen 4088 times before this occurrence. The second n-gram is the same but has been seen once more before its occurrence so has been seen 4089 times. The third n-gram has been seen 2300 times before and the last n-gram is seen for the first time.

To calculate the score for each log, we look at the n-grams they are present in. Below are the detailed calculations for the average count and the resulting anomaly score for each log:

First Log (Log 1):

- Present in n-gram 1.
- Counts: 4088.
- Average count:

$$\bar{C}(L_1) = \frac{\sum_{N_k \in N(L_1)} C_L(N_k)}{|N(L_1)|} = \frac{4088}{1} = 4088$$

- Anomaly score:
$$S(L_1) = \frac{1}{4088} = 0.0002$$

Second Log (Log 2):

- Present in n-gram 1 and n-gram 2.
- Counts: $4088, 4089$.
- Average count:
$$\bar{C}(L_2) = \frac{\sum_{N_k \in N(L_2)} C_L(N_k)}{|N(L_2)|} = \frac{4088 + 4089}{2} = \frac{8177}{2} = 4088.5$$
- Anomaly score:
$$S(L_2) = \frac{1}{4088.5} = 0.00024414$$

Third Log (Log 3):

- Present in n-gram 1, n-gram 2, and n-gram 3.
- Counts: $4088, 4089, 2300$.
- Average count:
$$\bar{C}(L_3) = \frac{\sum_{N_k \in N(L_3)} C_L(N_k)}{|N(L_3)|} = \frac{4088 + 4089 + 2300}{3} = \frac{10477}{3} \approx 3492.33$$
- Anomaly score:
$$S(L_3) = \frac{1}{3492.33} \approx 0.00028641$$

Fourth Log (Log 4):

- Present in n-gram 2, n-gram 3, and n-gram 4.
- Counts: $4089, 2300, 1$.
- Average count:
$$\bar{C}(L_4) = \frac{\sum_{N_k \in N(L_4)} C_L(N_k)}{|N(L_4)|} = \frac{4089 + 2300 + 1}{3} = \frac{6389}{3} \approx 2129.67$$
- Anomaly score:
$$S(L_4) = \frac{1}{2129.67} \approx 0.00046949$$

Fifth Log (Log 5):

- Present in n-gram 3 and n-gram 4.
- Counts: $2300, 1$.
- Average count:
$$\bar{C}(L_5) = \frac{\sum_{N_k \in N(L_5)} C_L(N_k)}{|N(L_5)|} = \frac{2300 + 1}{2} = \frac{2301}{2} = 1150.5$$
- Anomaly score:
$$S(L_5) = \frac{1}{1150.5} \approx 0.00086831$$

Sixth Log (Log 6):

- Present in n-gram 4.
- Count: 1.
- Average count:
$$\bar{C}(L_6) = \frac{\sum_{N_k \in N(L_6)} C_L(N_k)}{|N(L_6)|} = 1$$
- Anomaly score:
$$S(L_6) = \frac{1}{1} = 1$$

For the experiment, an n-gram size of 5 was employed for BGL, as it allows for the computation of anomaly scores for groups with a small number of logs but still utilizes the context around the log. For the HDFS dataset an n-gram of size 3 was employed as some of the anomalous blocks in the dataset are only 3 or 4 logs in total, which would result in no scores for that block. Additionally, a stride of 1 was utilized to maximize the number of data points available for each log.

5.5. FlexFringe

The other anomaly detection method utilized in this study is FlexFringe. It is a state-of-the-art technique that can be used for anomaly detection. It is used in this research to see what the effect of views is on a different type of anomaly detection than just the n-grams, which are a rather simple anomaly detection method. FlexFringe uses the log keys within a sequence of logs to construct a finite state automaton using a state-of-the-art evidence-driven state-merging algorithm. With this model, given a sequence of logs, FlexFringe can predict the state sequence of a given log sequence and provide the probability of that state sequence occurring. Further details on the implementation and capabilities of FlexFringe can be found in [14]. It is important to note that the FlexFringe settings employed in this study involve creating a state machine with an initial node based on sliding windows of a specified size and stride.

To detect anomalies, FlexFringe is provided with the entire dataset to learn a state machine. This process uses a sliding window size of n. The stride is set to 1 to maximize the number of state predictions per log and to be able to obtain state sequences within a small group. Subsequently, the state sequence is predicted using the same dataset. For each log, the states it reaches are recorded, along with the total number of times each state is reached. To compute an anomaly score for each log, 1 is divided by the average count of the states reached by the log. If a log reaches states that are rarely visited by other logs, it is considered anomalous.

When creating a view, the data is grouped while maintaining the sequential order within each group. A state machine is learned on that data, and the same data is used to predict the state sequences. Only the state sequences that are strictly within a group are used to extract the states recorded for each log.

The configuration for the FlexFringe models used in the experiment are as follows:

- heuristic_name=”alergia”
- data_name=”alergia_data”
- slidingwindow=1
- sinkson=1
- swsize=4
- swstride=1

- sinkcount=200
- printblue=1
- swaddshorter=1

A sliding window of 4 were chosen for this research but sliding windows shorter than that are also added. This is a much smaller size than what would be optimal for creating state machines but the runtime of bigger sliding windows would be to long to run on a student laptop. Sinks were used as it speeds up the runtime of the model. Sinks are states with user-defined conditions that are ignored by the merging algorithm. In this case the amount of visits to a state has to be higher than 200 for it to be merged with other states. We do however still want those states to be part of the state machine as they might indicate an anomalous state. Therefor we set printblue to true as this keeps sink states in the machine instead of removing them.

5.6. Finding Interesting Words

To better understand potential anomalous behavior in the data, we can focus on identifying specific words that frequently appear in logs exhibiting new behavior, but are rare in logs showing common behavior. This approach helps pinpoint terms associated with unusual activity. However, a limitation of the anomaly detection methods used in this research is that they primarily detect the beginning of brute force or DDoS attacks. Once such activities become frequent, they are categorized as normal behavior. Thus, words associated with these attacks might not seem anomalous if evaluated based solely on their average anomaly score.

To address this issue, the frequency of appearances of words in logs that show behaviour that is seen for the first time is stored, while excluding words that appear in a high percentage of total logs, such as ’a’ or ’system’. Specifically, words that occur in more than 20% of all logs are filtered out, because they might appear frequently in anomalous logs simply by chance rather than because the words themselves are indicators of new or anomalous behaviour.

Words are identified by finding strings separated by spaces in a log. These words are then cleaned by removing symbols attached to them, such as ”:”, ”(”, ”)” and ”’”. This prevents the scores of a single word from being split over two words. For example, the log message: ”Computer 1a has got a new logon. User:Pieter, Computer:1a”. First gets split into words seperated by a space, but User:Pieter and Computer:1a are then still seen as a single word. With the removal of the symbols they are turned into the words User, Pieter, Computer and 1a.

The anomaly score of the log is recorded for each word present in the log. For the first logs in the data, the anomaly score is always high due to the lack of known behavior patterns. Therefore, the scores of the first 5% of logs are not stored. The final score lists are then used to calculate the anomaly score for each word.

5.7. Finding Anomalous Users

The datasets from APTA and EYE contain data from servers where an attack has occurred. These attacks are carried out by users on the system, providing a clear indication of where to focus within the data. However, given that there can be up to 1452 users on the system, identifying the malicious user amidst those engaged in normal activities can be challenging.

To determine which users might be anomalous, we focus on the behavior exhibited by each user. It is more insightful to compare the behavior of a user to that of other users, rather than to the regular behavior of the entire system.

To facilitate the comparison of users, all logs associated with a user are collected using the following regular expressions, where 'xxx' represents a potential username:

1. Username: xxx
2. xxx@xxx
3. C:/Users/xxx/

Once the logs are identified, n-grams are generated from the logs. Using the n-grams for each user, an event-count matrix is constructed. In this matrix, each row corresponds to a user, each column represents an n-gram, and the value at row i and column j indicates the frequency of n-gram j in the logs of user i. The similarity scores between the rows of the matrix are then calculated using cosine similarity, and the average similarity score is used to identify users who deviate from the norm. Users are ranked based on how different their behavior is from other users, and this list can be utilized by analysts to identify potentially anomalous users. Users with fewer than 50 logs are excluded from the analysis because they have too few logs to have likely performed an attack and often do not represent normal user behavior. Including these users would only introduce noise into the technique, and therefore, they are left out to maintain the accuracy and reliability.

The results consist of a list of users with a similarity score, where a lower score indicates a greater deviation from other users. The most anomalous users should be prioritized for further investigation. Among the most anomalous users, there are often non-users who are incorrectly identified by the regular expressions used. These non-users rank high in the list of anomalous users because they do not exhibit typical user behavior. Additionally, some non-anomalous users, such as admins, may appear very different from regular users due to their unique activities.

To quickly identify which users are actually anomalous, it is helpful to examine the time range of their logs. It is often known when the attack occurred. Users who have been behaving consistently for years or months before the attack and continue to do so during the attack are likely regular users. In contrast, users who suddenly start behaving differently may indicate anomalous activity. Particularly interesting are users who are only present around the time of the attack and are very active during that period; these users should be investigated thoroughly.

A great way to review the logs of the users is through the dashboard described in section 5.8.3. The dashboard allows filtering logs by user and enables zooming in on specific timeframes to examine the details of each log entry.

This method effectively mitigates the impact of brute force attacks or other types of attacks that involve repetitive actions. Typically, such behavior might be considered normal due to its frequency. However, if these attacks are perpetrated by only a few users, this method highlights such behavior as anomalous, even though it may be common within the system as a whole.

5.8. Evaluation

5.8.1. Datasets

This section discusses the four datasets are used for the evaluation of the results. The first being the BGL dataset, the second is the HDFS dataset, the third is the APTA dataset and the last one the EYE dataset. The BGL and HDFS datasets are chosen as they are the most used datasets in research that are labeled and that contain software logs. The APTA en EYE datasets are chosen as it is a dataset required from the industry which makes it a more useful research for the industry than if the research was done on only research data, which is often clean and 'ready to use'.

The BGL dataset is discussed in [11]. This dataset is part of LogHub, a collection of system log datasets, described in [19] and consists of system logs of a super computer. The dataset contains a total of 4,747,963 logs from a supercomputer that are labeled as benign or with a type of anomaly. The logs contain a timestamp, date, node, type, component and the content of the log. The whole dataset is used for the evaluation.

The HDFS dataset, [17], contains of system logs generated by the Hadoop Distributed computing network. In total there are 11,175,629 logs divided over 575062 blocks that are labeled as normal or anomaly. 16838 blocks are anomalies. To limit the run time of the algorithms used, only the first 5,000,000 logs are used.

The APTA dataset exists of evtx logs from different servers. The data comes from real world data and is not as clean as data that is often used in research. In total there are CENSORED logs. With a brute force attack from an anomalous user. The data is not labeled but we know that there is an attack happening at the CENSORED

The EYE dataset is very similar to the APTA dataset however there are CENSORED logs in total and the attack is also performed by an anomalous user. The data is again not labeled but the attack happened on the CENSORED.

5.8.2. Timed Windows

Although the anomaly scores are assigned to individual logs, the evaluation is not conducted on single logs. This is because, in real-world scenarios, the crucial aspect is not necessarily classifying every single log correctly, but rather raising an alarm when something anomalous occurs. Whether this alarm is triggered precisely at the anomalous logs or within their vicinity is of lesser importance. Therefore, time windows are employed to categorize a cluster of logs within a specific timeframe as anomalous or not. The chosen timeframe is based on the dataset used. For the BGL dataset, given that the dataset comprises logs from a supercomputer where numerous logs are generated within a single second, 30 seconds is chosen as the timeframe. To establish an anomaly score for each window, the maximum anomaly score within the window is used as the anomaly score for that window, and a window is seen as anomalous if at least one log within it is deemed anomalous.

Since the HDFS dataset is labeled per block-ID, the highest score of the logs within a block will be taken as the score for that block.

To assess the predictions of the time windows, two real-life scenarios are considered. Firstly, proactive anomaly detection, where the detection method is employed to identify anomalies while a system is operational. In this context, minimizing false positives is crucial, as each false positive necessitates human intervention to determine if an anomaly truly exists. Secondly, incident response. In incident response, achieving a high true positive rate is essential, as analysts must identify the cause of an incident. Missing an anomaly in the data could lead to the analyst being unable to uncover the underlying anomaly. These scenarios are considered to make the results useful for not only the research field but also the real world industry.

To accommodate both scenarios, the Area Under the Curve (AUC) is utilized to evaluate the anomaly detection methods. AUC illustrates the true positive and false positive rates for various thresholds regarding whether a window is deemed anomalous. This enables assessment of performance concerning achieving a high true positive rate, low false positive rate, or a combination of both.

5.8.3. Manual Inspection

Many datasets lack labeled data because labeling logs, especially when there are millions of them, requires a significant amount of effort. However, it is still possible to manually check whether predictions are accurate. The downside of this manual checking is that it does not provide numerical results for easy comparison.

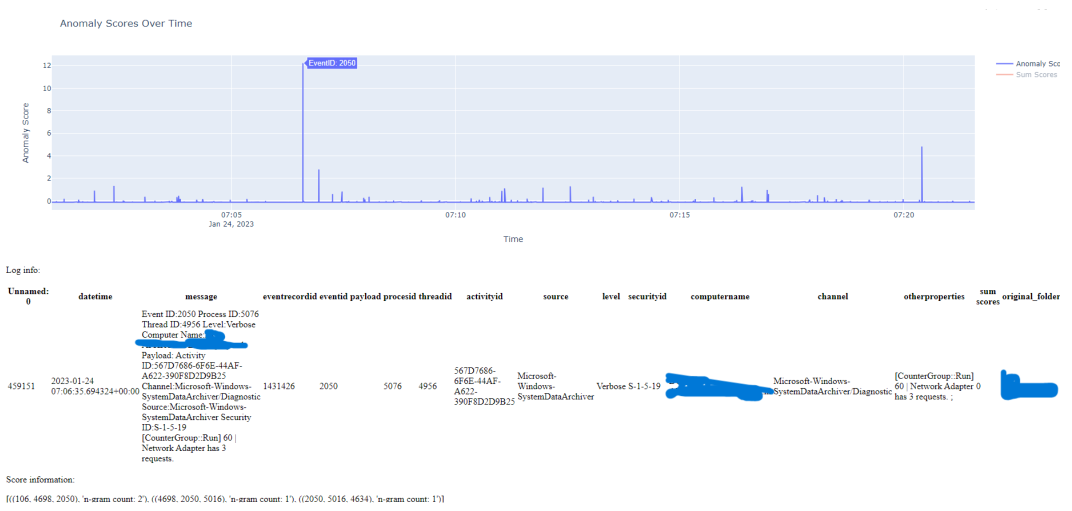
For the APTA data, it is known that an attack occurs at a specific time, carried out by an anomalous user, but the exact anomalous logs are not identified. Using the knowledge of the attack, it is possible to verify whether a technique detects unusual activity around that time. Additionally, since the logs before the attack are known to be benign, there should be minimal high anomaly scores for those logs.

To facilitate manual inspection, a dashboard was created to display anomaly scores over time. This dashboard allows users to zoom in on specific timeframes and adjust which data points are shown in the graph. For instance, users can select a specific user, and the logs containing that user's activity will be highlighted in the graph. Additionally, it is possible to search for certain words of interest or only show logs from a certain server.

The dashboard provides a detailed view when hovering over a data point, showing all the information about that log, including the calculation of the anomaly score. This feature makes it clear why a particular log has a high score, enhancing the interpretability of the anomaly detection process.

Not only is the dashboard useful for evaluation of the results of anomaly detection, it can also be used by analysts to find anomalous behaviour in their dataset. It allows them to zoom in on suspicious areas, find users or components that are anomalous and see all the logs that contain these users or components.

An example of the dashboard can be seen in figure 5.2.



Log info:

Unnamed: 0	datetime	message	eventrecordid	eventid	payload	procesid	threadid	activityid	source	level	securityid	computername	channel	otherproperties	sum scores	original_folder
459151	2023-01-24 07:06:35.694324+00:00	Event ID:2050 Process ID:5076 Thread ID:4956 Level:Verbose Computer Name:[redacted] Payload: Activity ID:567D7686-6F6E-44AF-A622-390F8D2D9B25 Channel:Microsoft-Windows-SystemDataArchiver/Diagnostic Source:Microsoft-Windows-SystemDataArchiver Security ID:S-1-5-19 [CounterGroup::Run] 60 | Network Adapter has 3 requests.	1431426	2050		5076	4956	567D7686-6F6E-44AF-A622-390F8D2D9B25	Microsoft-Windows-SystemDataArchiver	Verbose	S-1-5-19	[redacted]	Microsoft-Windows-SystemDataArchiver/Diagnostic	[CounterGroup::Run] 60 | Network Adapter 0 has 3 requests. ;		[redacted]

Score information:

[((106, 4698, 2050), 'n-gram count: 2'), ((4698, 2050, 5016), 'n-gram count: 1'), ((2050, 5016, 4634), 'n-gram count: 1')]

Figure 5.2: Example of the dashboard.

The figure displays a graph with anomaly scores over time. Below the graph, there is a table containing the columns of the selected log from the graph, along with their corresponding values. Underneath the table, the n-grams that the log was part of are listed together with the amount of times that n-gram was seen before the log.

5.8.4. Baseline Models

To better understand the results of the anomaly detection techniques used in the research, two baselines models are used.

To better understand the results of the anomaly detection techniques used in the research, two baseline models are employed.

The first baseline is PCA using event count matrices based on timed windows. First the logs are divided into windows of a certain time frame, for instance all logs within 5 minutes. These time frames are then used to create an event count matrix. Each time frame corresponds to a row in the matrix, and each column representing an event-ID. The value at row i and column j indicates the number of times event-ID j appears in time frame i. PCA is then applied to the event count matrix, retaining the minimum number of components needed to explain 95% of the variance. This approach ensures that non-anomalies are well reconstructed, while anomalies, which cannot be reconstructed as accurately, will have a higher reconstruction loss. The reconstruction loss is used as the anomaly score for the time windows.

PCA uses the event-IDs in the entire window and makes a prediction for the whole window. While the predictions for these windows are often accurate, a significant downside is that scores are calculated for the entire window, without indicating which specific logs might be anomalies. This can help analysts by highlighting anomalous areas but does not assist in pinpointing exactly which logs could be anomalous. An example of PCA can be found in section 2.5.

The second baseline simply counts how often the event-ID of a log has been seen so far. Only the event count of that log is used so no context is used at all. This baseline is chosen to evaluate what the scores would be if no context were used at all, relying solely on the rarity of the log key to determine the anomaly score.

6 Results

This chapter will discuss the results of the research. Each section represents a research question. Within research question 2 the results of finding interesting words and anomalous users are also shown.

6.1. Research Question 1

For the results of research question 1, we will primarily examine the differences between the two curves for the n-gram or FlexFringe methods. This comparison allows us to understand the impact of using different views on the same data this comparison will be made first. After that, the two baseline models will be utilized to assess the importance of context by comparing the results with the event-ID count and to evaluate the overall effectiveness of the technique used against other widely used techniques we will compare the methods to PCA. The n-grams and FlexFringe results are in 2 different plots to make the plots less cluttered.

The results of predicting the BGL dataset using n-grams are depicted in Figure 6.1.

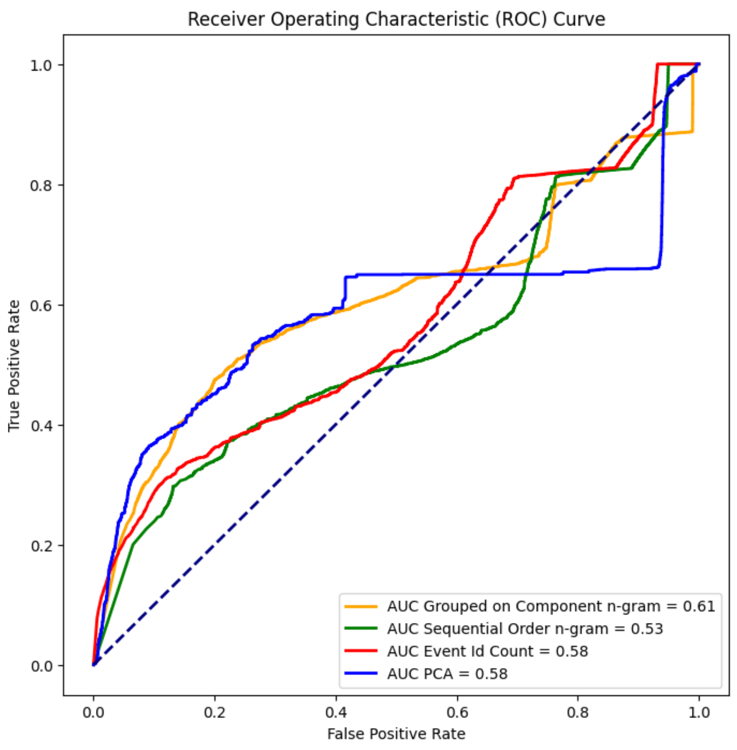


Figure 6.1: AUC-curve for the BGL dataset using n-grams

The figure shows the Receiver Operating Characteristic (ROC) curves for four methods: n-grams with both sequential data and data grouped on component, PCA on timed windows and predictions on logs based on event counts. All curves follow a similar trend, but there are distinct differences in their performance across various ranges of the false positive rate (FPR) and true positive rate (TPR). First we will discuss the differences between the two n-gram techniques, then we will compare the techniques to the baselines.

The curve for data grouped by component (orange) performs better at lower values of the FPR than curve for for sequential data. This indicates that this method is more effective at maintaining a lower rate of false positives while still achieving relatively high true positive rates. A good example of how grouping data by component helps identify anomalies that are missed when using sequential data is when the same error repeatedly occurred in the data on different processes. This for instance happens at the last few logs of the BGL dataset. The last few logs are the following:

Line-ID	Label	Component	Log Message	Event-ID
4747957	KERNSOCK	R31-M1-NC-I:J18-U11	FATAL idoproxy communication failure: socket closed	ab35a286
4747958	KERNSOCK	R20-M1-NC-I:J18-U11	FATAL idoproxy communication failure: socket closed	ab35a286
4747959	KERNSOCK	R00-M0-NC-I:J18-U11	FATAL idoproxy communication failure: socket closed	ab35a286
4747960	KERNSOCK	R36-M0-NC-I:J18-U11	FATAL idoproxy communication failure: socket closed	ab35a286
4747961	KERNSOCK	R30-M0-NC-I:J18-U11	FATAL idoproxy communication failure: socket closed	ab35a286
4747962	KERNSOCK	R31-M0-NC-I:J18-U11	FATAL idoproxy communication failure: socket closed	ab35a286
4747963	KERNSOCK	R34-M0-NC-I:J18-U11	FATAL idoproxy communication failure: socket closed	ab35a286

In sequential data, this repetitive error was eventually be perceived as normal behavior as it had been seen before. The n-gram counts for the sequential data were the following:

Line-ID	N-gram Counts
4747957	(7d9df963, 4bbe7086, 4bbe7086, ab35a286, ab35a286): 1 (4bbe7086, 4bbe7086, ab35a286, ab35a286, ab35a286): 1 (4bbe7086, ab35a286, ab35a286, ab35a286, ab35a286): 2 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 123 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 124
4747958	(4bbe7086, 4bbe7086, ab35a286, ab35a286, ab35a286): 1 (4bbe7086, ab35a286, ab35a286, ab35a286, ab35a286): 2 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 123 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 124 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 125
4747959	(4bbe7086, ab35a286, ab35a286, ab35a286, ab35a286): 2 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 123 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 124 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 125 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 126
4747960	(ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 123 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 124 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 125 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 126
4747961	(ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 124 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 125 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 126
4747962	(ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 125 (ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 126
4747963	(ab35a286, ab35a286, ab35a286, ab35a286, ab35a286): 126

However, when data is grouped by component, the normal behavior of a component is contrasted with an unexpected log. Since the same error occurs across different components performing different actions, it is not immediately perceived as normal behavior. This difference enables the identification of the anomaly that sequential data might overlook. The N-gram counts for data grouped on component is the following:

Line-ID	N-gram Counts
4747957	(7d9df963, 4bbe7086, 4bbe7086, ab35a286, ab35a286): 1
4747958	(29cbf9c8, 29cbf9c8, efb0cb55, 2558fcfa, ab35a286): 1
4747959	(efb0cb55, efb0cb55, 29cbf9c8, 2558fcfa, ab35a286): 1
4747960	(efb0cb55, 29cbf9c8, 29cbf9c8, 2558fcfa, ab35a286): 1
4747961	(2558fcfa, efb0cb55, efb0cb55, 29cbf9c8, ab35a286): 1
4747962	(efb0cb55, efb0cb55, efb0cb55, 4513d091, ab35a286): 1
4747963	(efb0cb55, 29cbf9c8, 4513d091, 2558fcfa, ab35a286): 1

On the other hand, the curve for sequential data (green) performs slightly better at higher TPRs, which suggests that it is more effective in identifying true positives when the FPR is higher. A significant part of this can be explained by the fact that some components do not have enough logs to create n-grams from. For example, there are a few logs without a component assigned to them. These logs are combined but have fewer than 5 entries, preventing the creation of n-grams. Consequently, they receive a score of 0, even though they are labeled as anomalous. There are several similar cases where other anomalous components have very few logs. This explains why the n-gram method grouped by component exhibits a relatively flat line on the ROC curve from an FPR of 0.85 to an FPR of 1.

In the context of incident response, where achieving a high TPR is crucial, the sequential data method might be slightly preferred due to its better performance at higher TPRs. However, for proactive anomaly detection or overall performance, the data grouped by component method demonstrates superiority. This is evident as it maintains higher TPRs across various low FPRs and achieves a higher Area Under the Curve (AUC) value of 0.61, compared to 0.56 for the sequential data. The higher AUC value indicates better overall performance in distinguishing between normal and anomalous behavior.

The context of the logs, defined by the sequence of other logs occurring at the same time, can be crucial when making predictions on the BGL dataset. This is evident as the event-ID count curve is overshadowed by the n-gram technique curve grouped by component at lower values of false positives. While PCA performs well at lower false positive rates, it performs poorly at higher positive rates. This discrepancy may be due to PCA's focus on entire time windows, where a single anomalous log may not significantly impact the window and thus go unnoticed. In contrast, the n-gram technique demonstrates robust performance against PCA on the BGL dataset, showing higher overall effectiveness with a higher AUC score of 0.61 for the grouped data compared to an AUC score of 0.58 for PCA. The simple event-ID counter method also underperforms compared to the n-gram method grouped by component. This might be due to its lack of context surrounding the logs, which could lead to missing important patterns. However, it does perform better than the n-grams on sequential data. This is likely because the sequential context of the logs may be irrelevant; the surrounding logs in a sequential context can pertain to entirely different processes, causing false positives and making it more difficult to identify actual patterns in the data.

The results of predicting the HDFS dataset using n-grams are depicted in Figure 6.2.

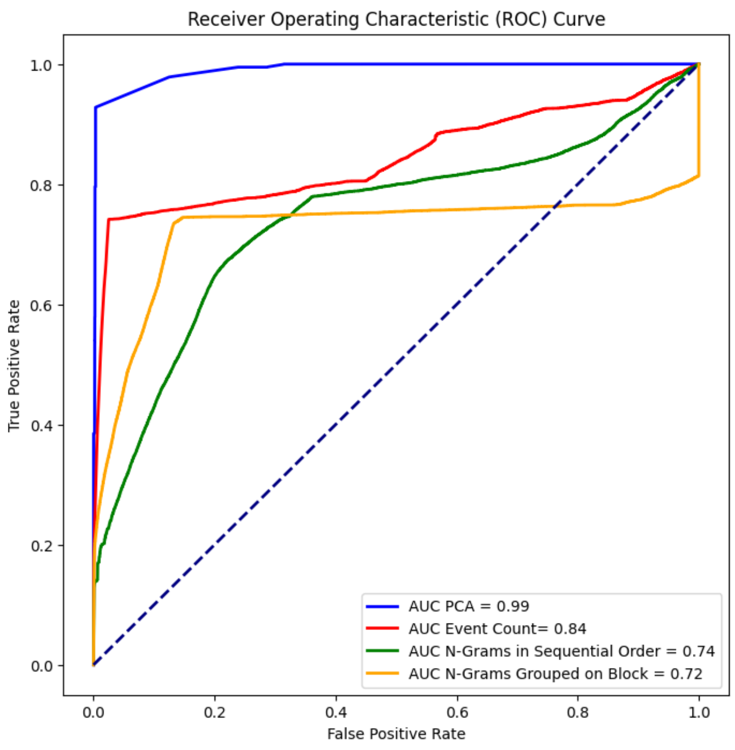


Figure 6.2: AUC-curve for the HDFS dataset using n-grams

The figure again presents the ROC curves for the same four methods. As with figure 6.1, both curves follow a similar trend, with distinct differences in performance across various ranges of the false positive rate (FPR) and true positive rate (TPR).

The curve for data grouped by block (orange) again demonstrates strong performance at lower FPR values, effectively maintaining a low rate of false positives while achieving relatively high true positive rates. Conversely, the curve for sequential data (green) shows slightly better overall performance, particularly at higher TPRs. This suggests that the sequential data method is more efficient at identifying true positives as the FPR increases.

In line with the previous analysis, for incident response scenarios where a high TPR is crucial, the sequential data method might be preferred due to its superior performance at higher TPRs.

However, for proactive anomaly detection or overall performance, the data grouped by block method still holds considerable merit, especially at lower FPRs. It maintains a strong balance between detecting true positives and minimizing false positives, similar to its performance in the earlier graph.

For this dataset, the sequential dataset performs slightly better overall with its higher Area Under the Curve (AUC) value of 0.74, compared to 0.72 for the data grouped by block. This higher AUC indicates better overall performance in distinguishing between normal and anomalous behavior.

However, the results indicate that both detection methods are inferior to a simple event count and PCA on the blocks. This outcome is expected, as anomalous blocks can be identified by the absence of an event-ID within a block or by the presence of very rare event-IDs, as discussed in section 2.7. PCA considers all event-IDs within an entire block, enabling it to detect blocks that are missing a specific event-ID or that contain rare event-IDs. Similarly, the simple event-ID counter can easily identify logs that are very rare. While n-grams can also detect rare logs, they can sometimes produce false positives when the order of common event-IDs appears for the first time, resulting in a high anomaly score even though the order may not be significant in this context.

The results of using FlexFringe on the BGL dataset can be found in figure 6.3.

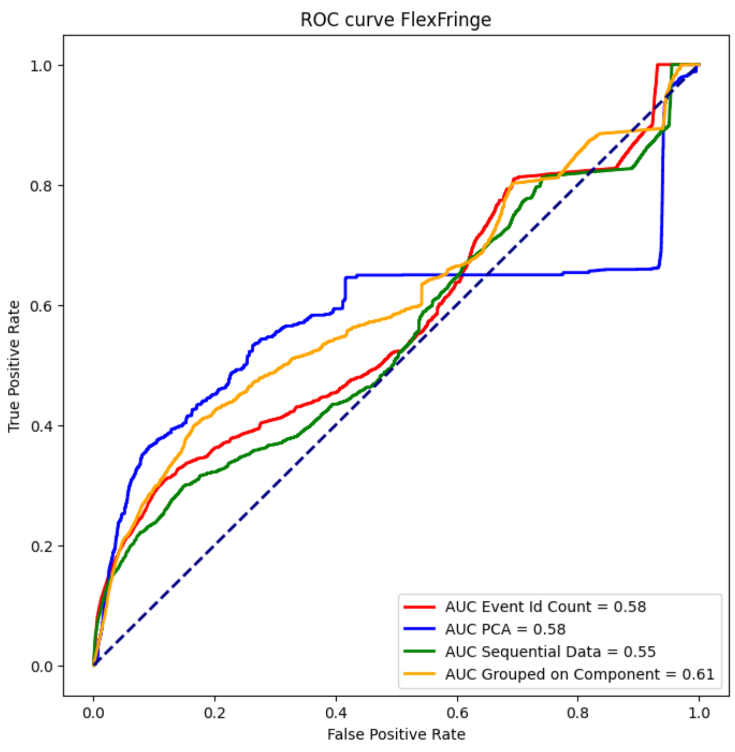


Figure 6.3: AUC-curve for the BGL dataset using FlexFringe

The grouped by component method (orange) performs slightly better at low FPR values than sequential data (green), indicating it is more effective at minimizing false positives while achieving reasonable true positive rates. From the midpoint onwards, the two methods perform similarly with still a slight adventage for the grouped method.

Overall, the grouped by block method works better than the sequential data method with AUC scores of 0.61 and 0.55 respectively. The difference is especially big at lower false positives rates but rather similar at high true positive rates.

This configuration of FlexFringe performs slightly better overall than PCA and the simple event-ID count method, achieving a higher AUC of 0.61 compared to their AUC values of 0.58. The reasoning for this is consistent with the explanation for Figure 6.1: PCA considers entire windows, where the impact of a single anomalous log is less noticeable. The event-ID counter method completely ignores the context of a log, potentially missing anomalous patterns. N-grams on sequential data perform the worst, likely due to using irrelevant context that results in false positives and makes it harder to detect meaningful patterns in the data.

The results of using FlexFringe on the HDFS dataset can be found in figure 6.4.

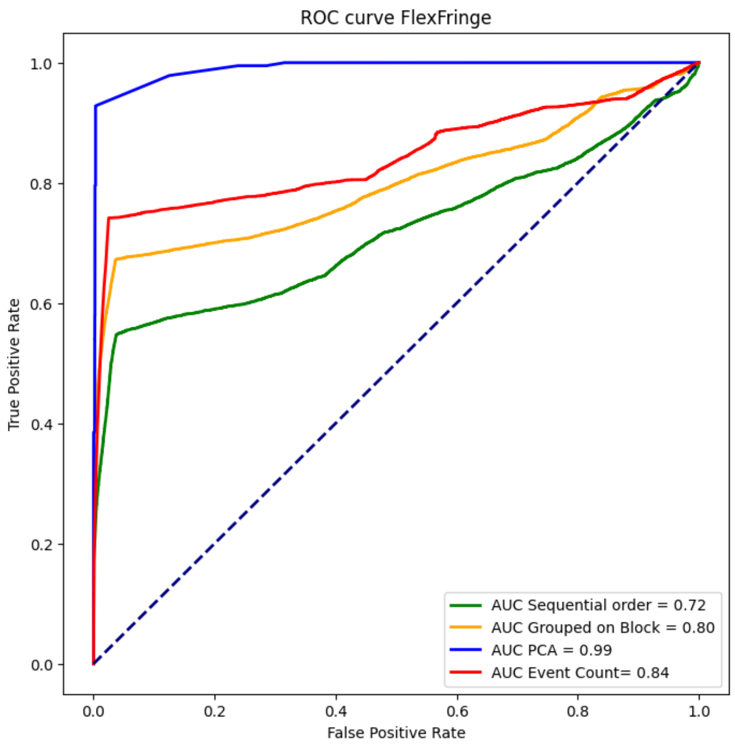


Figure 6.4: AUC-curve for the HDFS dataset using FlexFringe

In this graph, the performance between the different methods is very clearly visible as all methods follow a similar trend but at different performance. N-grams on sequential order (green) performs the worst with an AUC of 0.72 which n-grams based on block grouped data (orange) overshadows in every aspect with an AUC score of 0.80. FlexFringe does not have the problem of the input data being to short to create an anomaly score like n-grams have as it can also create predictions for input shorter than the sliding window.

It is clear that both the event-ID count and PCA outperform the n-gram methods and flexfringe methods on the HDFS dataset. This is due to the reason that are discussed in section 2.7 and in the explanation in figure 6.2: the anomalous blocks in the HDFS dataset can be identified well by missing event-IDS within the block, which PCA can recognize greatly, or by rare event-IDs that are easily picked up by the simple event-ID counter.

6.2. Research Question 2

To test the log grouping technique described in section 5.2, the HDFS dataset, as described in section 5.8.1, was used to find potential views. To obtain the log key and log values, the dataset was first parsed using DRAIN as described in section 5.1. The minimum overlap for cogenetic parameters was set to 0.9. In total 86 options are given. Many of which can be discarded because the log keys of the parameters are to specific caused by a mistake in the parsing. Such as:

Option 85:
First 10 values of 10 values: ['/10.250.10.176:50010', '/10.251.26.131:50010', '/10.251.75.79:50010', '/10.250.15.240:50010', '/10.251.71.68:50010',

*'/10.251.71.16:50010', '/10.251.123.1:50010', '/10.251.214.175:50010',
'/10.250.17.177:50010', '/10.251.203.80:50010']
on index 2 of log template : 10.250.10.223:50010:Transmitted block <*> to <*>
on index 2 of log template : 10.251.193.224:50010:Transmitted block <*> to <*>*

The ip-addresses are part of some of the log keys which is a mistake by the parser. These suggestions do show up last however as they contain very little amount of log keys and values.

Most options contain block-ids as values because block-ids are present in every log. The first options contains for instance 243,173 block-ids:

```
Option 0:
First 10 values of 243173 values: ['blk_-8400939786912755639',
'blk_8648024226389574687', 'blk_-7886599916563087367',
'blk_-5676752597777441092', 'blk_8879437951867746026',
'blk_3722814908549644161', 'blk_5399796255976554688',
'blk_-1029479317166673464', 'blk_-1553049055349049116',
'blk_-5737745528118688899']
on index 1 of log template : Received block <*> of size <*> from <*>
on index 1 of log template : 10.250.5.161:50010:Transmitted block <*> to <*>
on index 1 of log template : 10.251.195.70:50010:Transmitted block <*> to <*>
...
on index 1 of log template : PacketResponder <*> <*> Exception java.io.IOException:
Connection reset by peer
on index 1 of log template : 10.251.193.175:50010:Transmitted block <*> to <*>
on index 1 of log template : 10.251.66.102:50010:Transmitted block <*> to <*>
on index 1 of log template : 10.250.7.244:50010:Transmitted block <*> to <*>
on index 1 of log template : 10.251.105.189:50010:Transmitted block <*> to <*>
```

This option allows the program to look at the behaviour of blocks and might expose patterns that deviate from the expected behaviour.

The options that do not contain the block-ids as values, contain ip-addresses as values. This allows to see the data from a different perspective. One of the options with IP-addresses is option 11:

```
Option 11
First 10 values of 202 values: ['10.251.91.229:50010', '10.250.5.237:50010',
'10.251.91.159:50010', '10.251.90.64:50010', '10.251.71.68:50010',
'10.251.203.179:50010', '10.251.73.188:50010', '10.251.111.130:50010',
'10.251.42.84:50010', '10.251.195.52:50010']
on index 2 of log template :
BLOCK* NameSystem.delete: <*> is added to invalidSet of <*>
on index 3 of log template :
<*> writing block <*> to mirror <*>
on index 3 of log template :
<*> to transfer <*> to <*> got java.io.IOException: Connection reset by peer
on index 1 of log template :
<*> Starting thread to transfer block <*> to <*>
...
on index 2 of log template :
BLOCK* NameSystem.addStoredBlock: addStoredBlock request received for <*> on <*>
size <*> But it does not belong to any file.
on index 2 of log template :
BLOCK* NameSystem.addStoredBlock: Redundant addStoredBlock request received for
```

<> on <*> size <*>*

To create a view out of this option, the log-data is grouped by the ip-addresses and, in this case, anomaly detection using n-grams as explained in section 5.4 is used. The result for option 11 can be seen in figure 6.5.

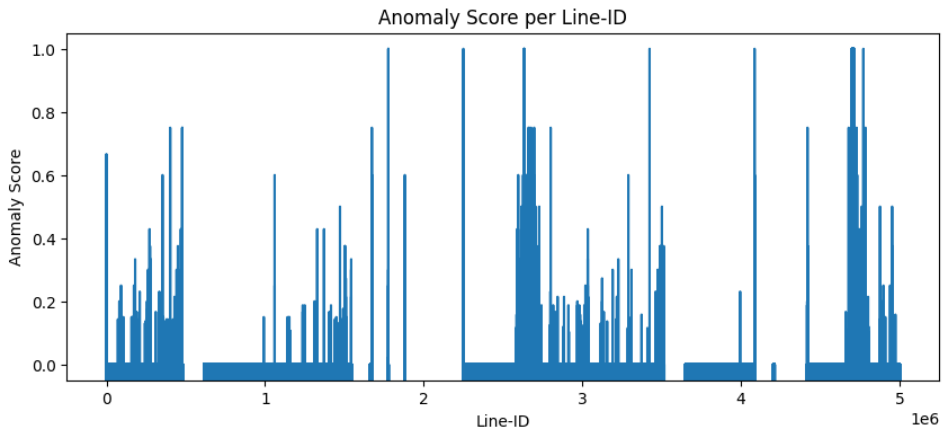


Figure 6.5: Anomaly Scores for IP-addresses view.

Logs that do not contain any ip-address are given a negative score so it is clear to see where there is a low anomaly score or no anomaly score. The data shows a view peaks of scores at points where the behaviour of an IP-address is different from the behaviour that is expected from an IP-addresses. This view does not contain enough anomaly scores on its own to do proper anomaly detection on all the data but it can however be used in combination with other views to detect anomalies based on different perspectives.

The last tactic used to create a view is to use domain knowledge to find groups in the data, as described in section 5.2. To do this we used the APTA dataset, together with the knowledge that there are users that might be interesting to look at. These users are presented with username@domain, such as user@hotmail. These are found in the data and then logs containing the same username are grouped together to do anomaly detection using n-grams. The results of the anomaly detection can be found in figure 6.6.

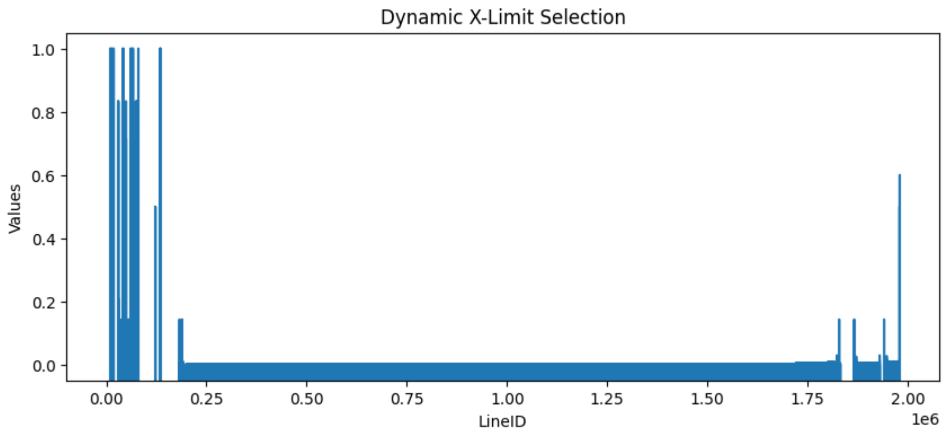


Figure 6.6: Anomaly Scores for users view.

The figure detects anomalous behaviour of users at the beginning and some at the end. Of the dataset we know that there is an attack happening at the end of the dataset. This however does not mean that the the behaviour at the beginning is not unexpected behaviour. For instance, if the system is updated, it introduces new behavior that deviates from the normal patterns in the data. It is only logical that this behavior is highlighted, as the goal is to identify behavior that is not typical. Detecting such deviations is crucial for ensuring that the anomaly detection system is effective in identifying both malicious activities and significant changes in system behavior.

6.2.1. Interesting Words

To better understand the APTA data, we looked for interesting words in the data that might show anomalous behaviour with the method explained in section 5.6. The results of running this method on the APTA data can be found in table 6.1.

Word	Anomaly Score	Total Number of Logs
at	8876.0	215448
context	8238.0	309356
root	8140.0	308534
Level	6157.0	165469
None	4872.0	60425
Name	3595.0	253488
is	3296.0	522002
cnt1	3033.0	35342
Int32	2913.0	31062
Verbose	2620.0	120010
String	2296.0	20587
The	2237.0	335267
Boolean	2140.0	21740
Active	2046.0	23209
server	2029.0	35299
Channel	1641.0	95142
to	1604.0	206661
Source	1595.0	181996
cnt2	1533.0	38356
Error	1520.0	31354
Account	1439.0	486936
not	1394.0	90137
partitionFqdn	1254.0	20920
CENSORED	1233.0	127464
Directory	1222.0	12085
#	1167.0	8199
Info	1143.0	3724
Application	1050.0	3211
message	980.0	12578
service	967.0	163986
cnt3	967.0	42959
callerFilePath	962.0	9250
callerFileLine	962.0	9250
memberName	962.0	9250
of	925.0	138600
operation	908.0	9864
Microsoft-Exchange-HighAvailability	903.0	1787
Object[]	896.0	5108
scope	872.0	5641
rootId	872.0	5640
a	817.0	391990
ChannelApplication	806.0	21229
CENSORED	772.0	35980
an	764.0	40881
—	756.0	26010
cnt4	732.0	40980
LDAP	730.0	11504
Logon	723.0	526362
directory	717.0	8666
Microsoft	708.0	27231

Table 6.1: Interesting Words in APTA Data

The table contains many non-interesting words such as 'at' and 'context'. However, there are some words that stand out and are worth examining more closely. For instance, the word 'LDAP' is notable because it is not a common word in regular logs. When filtering logs to only include those containing the word 'LDAP', it becomes evident that there are numerous anomalous logs during the attack period as can be seen in figure 6.7.

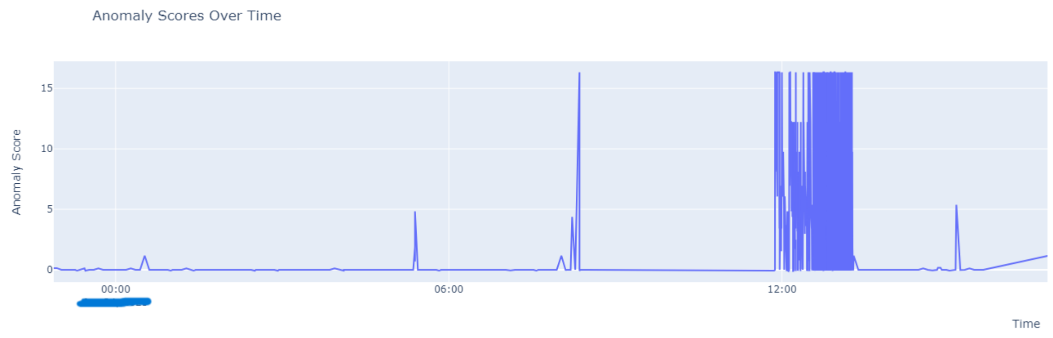


Figure 6.7: Anomaly Scores for the word 'LDAP'.

Logs containing 'LDAP' during the attack suggest that the server might have been out of service during the attack, providing an important clue for further investigation.

Another intriguing word is 'partitionFqdn'. This word appears exclusively in logs during the attack, as shown in figure 6.8, which was generated using the dashboard. This could help guide an analyst's investigation by highlighting specific events or conditions that occurred during the attack period, potentially pointing to the source of the attack.

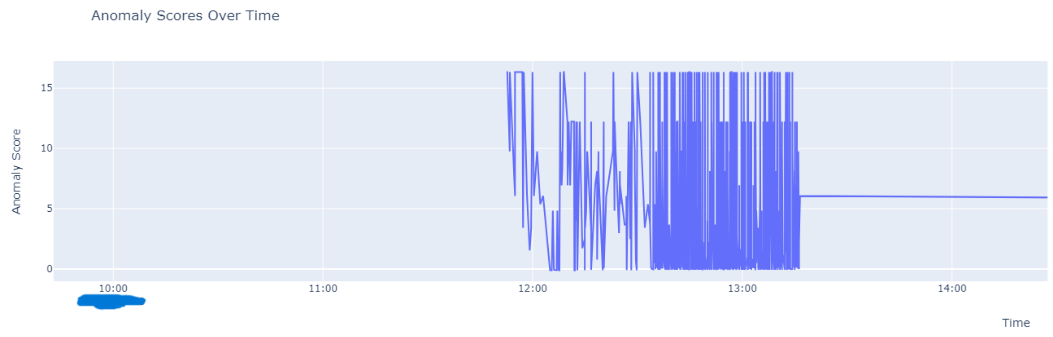


Figure 6.8: Anomaly Scores for the word 'partitionFqdn'.

The logs containing 'partitionFqdn' show that it is a variable name of a string of the function *Microsoft.Exchange.Data.Directory.ServiceTopologyProvider.GetConfigDCInfo(String partitionFqdn Boolean throwOnFailure)*.

6.2.2. Interesting Users

To identify the attack in the EYE dataset, the goal was to find users that might be anomalous, given that the attack was carried out by a user on the system. Therefore, the method explained in section 5.7 was applied to the data to determine if it was possible to identify the attacker. In total, 1453 users were detected in the data. The top 30 users with the lowest similarity scores are displayed in Table 6.2.

Rank	Similarity Score	Username
0	0.006622516556291387	name:
1	0.006622516556291389	name of a computer part with an @ in it
2	0.00662251655629139	domain name with an @ in it
3	0.006622516556291391	username1
4	0.006622516556291391	username2@CENSORED
5	0.0066225165562913925	nt
6	0.006733193816270513	admin.adminname
7	0.006971793147353612	-
8	0.013245033112582781	ready
9	0.013245033112582781	verify
10	0.013255832547730955	username3
11	0.013255832547730955	username4
12	0.026280899194660753	system
13	0.03310945724416292	username5@CENSORED
14	0.033112582781456956	computername1
15	0.033112582781456956	computername1.
16	0.033112582781456956	computername2
17	0.033112582781456956	computername2.
18	0.033112582781456956	computername3
19	0.033380014150051394	computername4
20	0.039460533890132196	username6@CENSORED
21	0.03962357192529833	variationofusername5@CENSORED
22	0.0406571339372369	username6@CENSORED
23	0.04181317936344641	computername5
24	0.042334607666605024	computername6
25	0.042809934624014105	computername7
26	0.04756266087142915	computername8
27	0.04874262031915246	computername9
28	0.04874262031915246	anomalous user
29	0.04874262031915246	username7
30	0.04963229606184337	computername10

Table 6.2: Similarity scores and usernames

The first three most anomalous 'users' are not actual users but were mistakenly detected as such. For example, the top-ranked 'user' contains an '@' symbol, leading the algorithm to misidentify it as a user, while it is actually part of a computer, making it unsurprising that it differs from other users. In fact, many of the 'users' in the top rankings are not actual users. Specifically, this applies to ranks 0-2, 5, 7-9, 12, 14-19, 23-27, and 30.

Ranks 4, 10, 11, and 27 are users who were active after the attack was detected and during the incident response, making their behavior different from other users. Rank 3 is associated with the user 'stratus,' who appears to be connected only to MySQL and is active only when there is an issue with MySQL. This has been a consistent pattern for years, indicating that this user is not an attacker but simply has unusual behavior. Rank 6 is an admin, whose activities differ from those of regular users, resulting in a high anomaly score.

Ranks 13, 20, 21, and 22 are very active users with a high volume of logs, distinguishing them from the average user, though they are not actually anomalous. Rank 28 is associated with the user 'anomalous user' whose logs are particularly suspicious. This user was only active for a very brief period, generating a large number of logs within a short timeframe just before an attack was noticed. The logs from this timeframe, as displayed on the dashboard, can be seen in Figure 6.9.

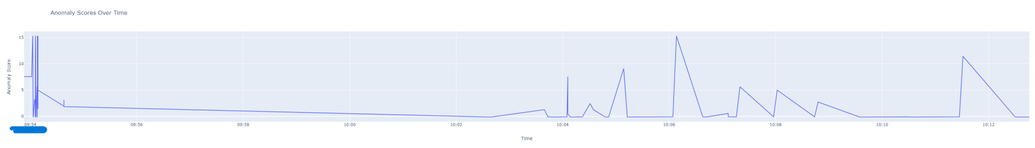


Figure 6.9: Logs of user anomalous user

Around 09:54, the user generates multiple logs per second. Then, at 10:02, the user creates a master key for the Microsoft-Windows-Crypto-DPAPI, which is used to encrypt and decrypt sensitive information. Afterwards, the user utilizes the Background Intelligent Transfer Service (BITS), a service known for uploading and downloading files and has been known to be exploited by attackers [9].

6.3. Research Question 3

To assess the effect of combining multiple views, the APTA dataset was utilized because it offers multiple interesting views, unlike the HDFS and BGL datasets, which have only two or three views. The anomaly scores from the different views were first normalized using z-score normalization. This normalization highlights how anomalous a log is within a specific view. It ensures that a log with a score of 1 in a view where many logs score 1, is considered less anomalous compared to a view where fewer logs have a score of 1, making the relative anomaly scores more apparent.

The anomaly scores for different views can be seen in Figure 6.10. The graph is zoomed in on the time of the attack.

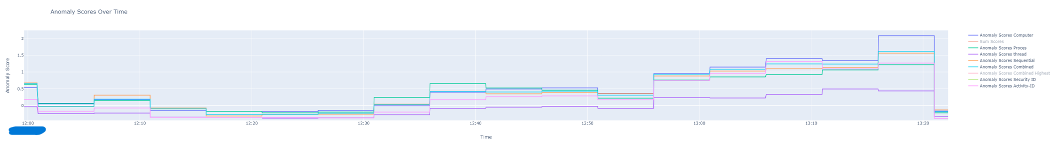


Figure 6.10: Anomaly Scores for different views.

The bulk of the attack occurs between 13:14 and 13:20. During this period, the view that groups data by computer name shows the highest anomaly score by a significant margin. This is followed by the combined view, which averages the scores of all other views, and is closely trailed by the sequential view. The thread-ID view is the least effective by a substantial margin.

While the combined view might not have the highest performance, it yields solid results compared to other views, such as the activity-ID or process-ID views. This is particularly useful because it is often unknown in advance where the attack might occur, making it challenging to predict which view will be the most effective. Using a combined view ensures a reliable performance, providing a safer option when the optimal view is not clear.

To determine if there is an optimal number for combining the nth highest scores, where n is the total number of views, the APTA data was applied to the method explained in section 5.3. The results are displayed using the dashboard, zoomed in on the attack window, as shown in Figure 6.11.

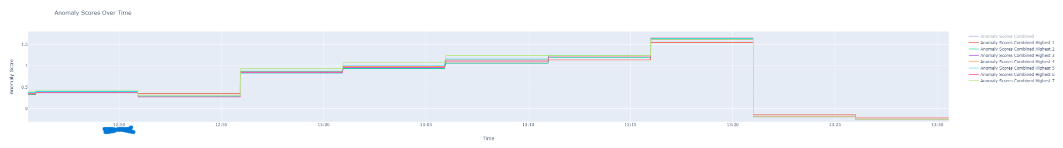


Figure 6.11: Anomaly Scores for the all options of the nth highest views.

The figure indicates that there is very little difference between the various options for combining the scores. The only noticeable difference is that the highest view score performs slightly worse than the other options. This is because the scores are normalized, and the high scores for this view during the attack window hold less significance due to the overall average score being much higher than for the other options.

Overall, it can be beneficial to combine views if it is not clear which views perform the best. This can be done by combining the highest x views or taking the average (which is x=number of views). Taking only the highest anomaly score of each view for a log can result in a lot of false positives and is therefor not recommended. Taking the average can always be a safe option that can be considered.

7 Conclusion

The graphs seen in section 6.1 show that the order in which the data is given to the anomaly detection method has influence on the results. The influence is dependent on the detection technique used and also depends on the dataset used. The chosen order should also keep in mind what the goal of the detection method is. One order could improve performance on low true positive value but decrease performance on high true positive rates. There might also be a difference between different orders used. Since the use of views can have a significant effect, it is good to consider if there might be useful views when using anomaly detection.

Views can be derived using various methods. The simplest approach involves using existing columns in the data, as these often contain important information and can provide some very useful views. Because of its simplicity and potential usefulness it should always be consider when present. However, there are instances where these columns may not yield the necessary insights or are not present at all, making it important to consider other methods for finding views.

Log grouping is another technique for identifying views, leveraging log keys and values. This method can create views based on values present in all logs or in subsets of logs. Views based on subsets may be less obvious and harder to find but can still offer valuable insights into the data. It does cost more time and effort to find views using log grouping but they may discover views that otherwise would not have been found and can be used without any knowledge of the data.

Domain knowledge is the final approach for creating views. This allows analysts to guide the anomaly detection method in the right direction and assist analysts in determining the usefulness of certain perspectives. A great example is the user similarity that was used on the APTA and EYE datasets. It was already known that users caused these attacks but not known which user performed the attack. By simply focusing purely on the users and comparing them against each other, the anomalous users were detected from a dataset with more than 1400 detected users and more than 7 million logs.

All three methods—using existing columns, log grouping, and domain knowledge—can yield useful views and should be considered when searching for the most informative perspectives on the data.

Identifying which view is particularly useful can be challenging when knowledge of the dataset is limited. Combining the results of multiple views can be a safe and effective approach, as it tends to perform as a middle ground among the views. Outliers, whether they are very high or very low scores within a single view, are balanced out by the other views. This ensures a more reliable outcome that performs around the average compared to all the views. However, the effectiveness of this approach still heavily depends on the relevance of the views combined; if all views are non-relevant, the combined results will still underperform.

Overall, this research has demonstrated the significant potential of using different views in anomaly detection. Their usefulness is highly dependent on the chosen view, the dataset, and the anomaly detection method employed. When used correctly, views can substantially improve results without requiring major adjustments to the existing anomaly detection framework and should therefor always be considered.

8 Limitations

While this study has provided valuable insights into the effect of views on the input data, it is important to recognize its limitations. By understanding these limitations, we can better interpret the findings and identify areas for future research. The following limitations will be discussed: the absence of repeated experiments, the parameters used for FlexFringe, the missing guarantee for useful views, the limited size of some groups within a view and the log key consistency needed to do log grouping.

The first limitation is the absence of repeated experiments on research question 3. The experiment was only performed on the APTA dataset. The BGL and HDFS datasets only had one or two interesting views in addition to the sequential view, compared to the six views in the APTA dataset. The EYE data is the same type as the APTA data but was not used due to time constraints.

Another limitation is the parameters used for FlexFringe. FlexFringe creates state machines from the input data. Due to hardware and runtime constraints, the sliding window size over the data was set to 4. This means the model was trained on an input string of 4 logs. This can make it much harder to learn a meaningful model. For instance, the length of a block in the HDFS dataset is between 13 and 31 for 95% of the blocks. Creating a model using all the logs from a block would likely result in a more meaningful model and better results.

The methodology itself also has some limitations. The most significant is that the usefulness of a view is not known beforehand without access to labeled training data. Often, software logs do not have labeled data, making it impossible to know for certain that a view will produce better results than sequential data. There are some methods discussed to determine which view might be interesting but this is no guarantee. With some knowledge about the dataset is is also possible to estimate how well a view will perform, but it is challenging to employ views on unknown datasets without significant effort.

Another limitation of views is the limited size that a component or group can have. For instance, in the HDFS dataset, there are many anomalous blocks that contain only 3 or 4 logs. For n-grams, this means that the n-gram size either has to be at most 3, or these blocks will not receive a score. This constraint limits the effectiveness of n-grams in detecting anomalies in smaller groups or components.

Another limitation is that log grouping might not work on all datasets. Log keys and values are needed to create parameters. Data with many different log keys but only a few different values does not work well with log grouping. For example, the EVTX logs of the APTA data have many different log keys as data columns are not constant, and the same event-ID might have a different log message. Using log grouping on that data did not result in any useful views

9 Future Work

The results highlight the substantial impact that views on the data can have, but this research is only a first step. There are multiple ways in which researchers can build upon this work.

The first approach is to apply views to already existing anomaly detection methods and replicate previous research using views as the input instead of the normal sequential data. This can demonstrate whether views can improve existing research and document how easily the anomaly detection methods can be adapted to use views. For example, recreating the study by Shilin He et al. [4] with views as the input could be insightful.

Another way to expand on views is by trying to combine multiple types of views. Instead of only combining views that generate scores for all logs, it could also be possible to integrate those views with views that give scores to a group themselves or a part of the logs. For instance with a user view that assesses how anomalous a user is. This might provide more context and explanation for why a log is considered anomalous.

Additionally, more information on why a log is anomalous can be derived by examining how the log deviates from previously observed behavior. For instance, if a log is part of several anomalous n-grams, it can be checked whether the log itself is very rare, causing the n-grams to be anomalous, or if the combination of this log with others is rare. If the combination is rare, it is possible to analyze what a normal combination with the log looks like to understand the normal behavior and what is currently happening.

Lastly, further development of the user similarity method is recommended. Currently, there is still a lot of manual work involved in verifying whether a user is truly anomalous. A simple check involves examining how long the user has been active. If a user is only active at the moment of the attack and then stops, they are likely anomalous. However, if a user has consistently exhibited behavior different from others for a long time, they are probably not anomalous. By analyzing the activity patterns of users, the algorithm might be able to reduce the number of false positives in the anomalous user list which reduces the manual work required.

References

[1] Qiang Fu et al. “Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis”. In: *2009 Ninth IEEE International Conference on Data Mining*. 2009, pp. 149–158. DOI: 10.1109/ICDM.2009.60.

[2] Hongcheng Guo et al. “LogFormer: A Pre-train and Tuning Pipeline for Log Anomaly Detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38 (Mar. 2024), pp. 135–143. DOI: 10.1609/aaai.v38i1.27764.

[3] Pinjia He et al. “Drain: An Online Log Parsing Approach with Fixed Depth Tree”. In: *2017 IEEE International Conference on Web Services (ICWS)*. 2017, pp. 33–40. DOI: 10.1109/ICWS.2017.13.

[4] Shilin He et al. “Experience Report: System Log Analysis for Anomaly Detection”. In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, pp. 207–218. DOI: 10.1109/ISSRE.2016.21.

[5] H. Hotelling. “Analysis of a complex of statistical variables into principal components.” In: *Journal of Educational Psychology* 24.6 (1933), pp. 417–441. DOI: 10.1037/h0071325.

[6] Zhong Li and Matthijs van Leeuwen. “Feature Selection for Fault Detection and Prediction based on Event Log Analysis”. In: *SIGKDD Explor. Newsl.* 24.2 (Dec. 2022), pp. 96–104. ISSN: 1931-0145. DOI: 10.1145/3575637.3575652. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/3575637.3575652.

[7] Zhong Li, Jiayang Shi, and Matthijs Leeuwen. “Graph Neural Networks based Log Anomaly Detection and Explanation”. In: May 2024, pp. 306–307. DOI: 10.1145/3639478.3643084.

[8] Jian-Guang Lou et al. “Mining invariants from console logs for system problem detection”. In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC’10. Boston, MA: USENIX Association, 2010, p. 24.

[9] Mandiant. *Attacker Use Background Intelligent Transfer Service (BITS)*. Mar. 2021. URL: https://cloud.google.com/blog/topics/threat-intelligence/attacker-use-of-windows-background-intelligent-transfer-service/.

[10] F.P.J. Nijweide. *Confidence is key: Using Gaussian Process Classifiers to improve robustness and interpretability of CNNs*. Jan. 2022. DOI: 10.13140/RG.2.2.31025.04966.

[11] Adam Oliner and Jon Stearley. “What Supercomputers Say: A Study of Five System Logs”. In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*. 2007, pp. 575–584. DOI: 10.1109/DSN.2007.103.

[12] Marcel Rumez et al. “Anomaly Detection for Automotive Diagnostic Applications Based on N-Grams”. In: *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2020, pp. 1423–1429. DOI: 10.1109/COMPSAC48688.2020.00-56.

[13] Akanksha Toshniwal, Kavi Mahesh, and R. Jayashree. “Overview of Anomaly Detection techniques in Machine Learning”. In: *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2020, pp. 808–815. DOI: 10.1109/I-SMAC49090.2020.9243329.

[14] Sicco Verwer and Christian Hammerschmidt. *FlexFringe: Modeling Software Behavior by Learning Probabilistic Automata*. 2023. arXiv: 2203.16331 [cs.LG].

[15] Xinjie Wei, Chang-ai Sun, and Xiao-Yi Zhang. “KAD: a knowledge formalization-based anomaly detection approach for distributed systems”. In: *Software Quality Journal* (May 2024), pp. 1–25. DOI: `10.1007/s11219-024-09670-8`.

[16] Rick Wieman et al. “An Experience Report on Applying Passive Learning in a Large-Scale Payment Company”. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2017, pp. 564–573. DOI: `10.1109/ICSME.2017.71`.

[17] Wei Xu et al. “Detecting large-scale system problems by mining console logs”. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP ’09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 117–132. ISBN: 9781605587523. DOI: `10.1145/1629575.1629587`. URL: `https://doi.org/10.1145/1629575.1629587`.

[18] Zhongjiang Yu et al. “LogMS: a multi-stage log anomaly detection method based on multi-source information fusion and probability label estimation”. In: *Frontiers in Physics* 12 (Apr. 2024). DOI: `10.3389/fphy.2024.1401857`.

[19] Jieming Zhu et al. *Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics*. 2023. arXiv: `2008.06448 [cs.SE]`.