Semi-supervised Configuration and Optimization of Anomaly Detection Algorithms on Log Data

Viktor Beck*, Max Landauer*, Markus Wurzenberger*, Florian Skopik* and Andreas Rauber†
*AIT Austrian Institute of Technology, Vienna, Austria
Email: firstname.lastname@ait.ac.at
†Vienna University of Technology, Vienna, Austria
Email: andreas.rauber@tuwien.ac.at

***Abstract*—Cyber threats are evolving rapidly, making anomaly detection (AD) in system log data increasingly important for detection of known and unknown attacks. The configuration of AD algorithms heavily depends on the data at hand. It often involves a complex feature selection process and the determination of parameters such as thresholds or window sizes. In many cases, configuration requires manual intervention by domain experts, which limits accessibility and effectiveness of AD algorithms. This work introduces a Configuration-Engine (CE), which employs a semi-supervised approach to automate the configuration process or optimize existing configurations. The CE utilizes statistical methods to identify log line properties to recognize meaningful tokens for AD methods to monitor. It categorizes variables by their characteristics and behavior over time, then specifies which log parts a detector should observe, and sets appropriate configuration parameters.**

The CE was evaluated using four different detectors. Evaluations on different Apache Access and audit datasets containing attack traces showed that the CE achieved an average precision of over 0.94 for Apache and over 0.79 for audit datasets, while maintaining high recall, competing with the performance of expert-crafted configurations. The optimization approach was able to strongly improve the precision of both the CE's and the experts' configurations for Apache data in 7 out of 16 cases. Furthermore, the CE's configurations were significantly dissimilar to each other when generated on audit data, highlighting the importance of automated configuration.

***Index Terms*—anomaly detection, log data, semi-supervised learning, feature selection, configuration generation**

I. Introduction

The landscape of cyber threats is constantly evolving, with novel attack techniques emerging at a rapid pace. Current reports such as the ENISA Threat Landscape Report 2023 [1] or the Crowdstrike 2024 Global Threat Report [2] name ransomware as the top cybersecurity threat. The rise of AI-enabled disinformation and supply chain attacks are also major concerns, as are persistent DDoS threats, phishing or social engineering used to gain initial access to systems. Moreover, attacks are becoming more targeted, with a focus on high-value sectors like manufacturing and industry. Especially, intrusions in cloud environments are strongly increasing. The dynamic nature of these threats presents a serious challenge to organisations, governments and individuals alike. It necessitates a continuous state of vigilance and the implementation of robust security protocols. Once an unauthorised party gains access or control of a system, they can exfiltrate or manipulate sensitive data, implant malware that is able to corrupt or destroy computer infrastructures, and disrupt critical services. Attacks can occur without detection by system administrators and the longer an attack persists unnoticed, the greater the potential damage. Consequently, early detection of potential intrusions is paramount to minimise risks and harm [2].

The majority of AD methods involve scanning for signatures such as hashes or IP addresses that correspond to known malware. As this approach only covers threats that have been observed and forensically analyzed before, it is not possible to detect intrusions based on new and unknown techniques [3]. Consequently, there has been a growing trend towards the use of data science methods, particularly AD, to address these challenges. These methods have gained popularity, because of their ability to detect system states that deviate from normal system behavior, thereby enabling identification of known and unknown intrusions [4].

Intrusion detection systems (IDS) learn the normal behavior of a system based on its log data to uncover possible intrusions by identifying anomalies. The configuration of AD algorithms requires customization based on the specific type of log data generated by the system, which varies depending on the system's expected events. This includes adjustment of thresholds, determination of parameters or feature selection. Log files sometimes contain millions of events, complicating the extraction of relevant information for effective configuration. The configuration of the detection tool is a non-trivial and tedious task and often carried out manually by domain experts.

The configuration determines which detectors the tool should employ and their corresponding parameter settings. Selecting the appropriate settings to adapt to the given context is critical to find anomalies while maintaining a low amount of false positives (FP). To illustrate this, an IDS with insufficient sensitivity may fail to detect anomalous instances corresponding to attacks with high accuracy, while one that is overly sensitive may generate numerous alerts for normal events along with intrusions. It would be unfeasible for administrators of large systems to distinguish between intrusions and non-hostile events [3], [4].

In order to address the above-mentioned challenges, this work presents the "Configuration-Engine" (CE), a method to automate the configuration of AD tools. The CE generates configurations from assumingly anomaly-free data, making

979-8-3503-6248-0/24/$31.00 ©2024 IEEE

it a semi-supervised approach. In simple terms, it assesses what aspects of the data are worth investigating and how to investigate them and passes this information to the detection system. The CE consists of a collection of configuration methods designed to effectively recognize various patterns in the data that represent some kind of learnable normal behavior. This approach involves a tradeoff, as each detector requires its own distinct configuration method rather than relying on a single model for the entire process. However, once the method is defined and the hyperparameters are optimized, it should be universally applicable. The CE was applied on and evaluated using four different detectors of the so-called Logdata-Anomaly-Miner, or AMiner, a modular pipeline for AD on log data [3]. The detectors are described in Sec. IV.

The remainder structures as follows: Section II provides an overview of the related work. Section III explains the concept and different steps of the CE. Section IV describes the detectors for which the configuration methods were exemplarily defined. Section V describes these configuration methods. Section VI describes how the feedback of the detectors applied on training data can be used to optimize configurations. Section VII describes which and how each configuration method can be meaningfully applied to which detector. Section VIII discusses the evaluation results. Section IX describes the limitations of the approach and future work and Sec. X concludes this paper.

II. Related Work

AD often utilizes distance metrics to assess if instances deviate from the norm. Typically, detection methods focus on quantitative features as deriving distance metrics for categorical data is challenging [5]. Inspiration for many methods, including mathematical definitions and the handling of categorical variables, comes from [6], particularly the use of co-occurrence for variable combinations (explained in Sec. V-D). They propose the Variable Correlation Detector (VCD), a detector that utilizes selection and filtering steps on categorical variables to reduce the search space for variable pairs to detect anomalies. Unlike frequent itemset mining techniques (association rule mining), the VCD retains infrequent variables valuable for AD. However, most association rule mining techniques are ill-suited for AD with log data due to the need for standardized formats and the volume and variability of log data in large-scale systems [7], [8].

In [9] Wurzenberger et al. emphasized the variable part of log lines, proposing an unsupervised approach called the Variable Type Detector (VTD) to classify log line tokens into data types such as static, acending, descending, unique and more. Anomalies are detected by changes in these data types. The concept of data types is also incorporated into this work.

The methods used here are primarily feature selection methods as it is critical to select input features for the AD algorithm with a high sensitivity to anomalies, in order to achieve high effectiveness [10]. Kloft et al. [10] therefore propose an automatic feature selection method using a generalized support vector data description model [11], while Pascoal et al. [12] suggest a method based on a mutual information metric and robust statistics.

A prominent anomaly detection approach for log data is DeepLog [13], a recurrent neural network (RNN) model that uses Long Short-Term Memory (LSTM) to detect anomalies in system logs. Other state-of-the-art approaches and similar to DeepLog are LogAnomaly [14], which also relies on LSTM RNN, and LogRobust [15], which focuses on robustness against unstable log data. Recent advances like LogBERT [16] and LogGPT [17] utilize the semantic capabilities of language models for anomaly detection.

Most publications assess their approaches using well-known datasets such as HDFS, BGL, Thunderbird, OpenStack or Hadoop which are publicly available on LogHub [18], [19]. These datasets became the standard to evaluate AD algorithms in literature, yet most of their anomalies can already be detected with simple machine learning techniques such as similarity-based clustering and do not require advanced techniques for detection [7]. Suitable datasets for evaluating AD approaches in the context of cyber security are AIT Log Data Set V1.0 [20] and AIT Log Data Set V2.0 [21], as their anomalies can often only be detected through complex analysis of event parameters. These datasets are therefore used for training, validating and testing the CE's implementation. Similarly to [20] and [21] the log data used for evaluation of [22] includes attacks such as SQL-injection or bruteforce login. Yet, [20], [21] constitutes a more extensive evaluation environment as they include several more attack types and sophisticated attack vectors, such as scans for vulnerabilities or open ports followed by webshell uploads and privilege escalation or data exfiltration and many more. The resulting anomalies are often non-trivial to detect.

In our previous work we proposed an incremental clustering approach for AD in [22] to enable clustering for online processing, i.e. processing data streams. We also proposed the AMiner [3] which is built for efficiently processing log lines in an online mode. The CE is inherently an offline approach, but processing log lines in a batch-wise manner would allow the usage in online AD frameworks. Processing data in real time requires efficient mechanisms. All standard parsers of the AMiner therefore utilize tree structures similar to Drain [23] for efficient parsing.

A very recent advance comes from [24], which also addresses the configuration problem for AD. They use network data with attacks that have been previously labeled in an offline phase that trains a meta-model in a supervised manner. Since it is supervised, it is able to define optimal configuration parameters for the AD algorithm. In an online phase, the newly obtained and presumably attack-free data is then used as input to the pre-trained meta-model, which outputs approximations to the optimal configuration parameters. While their approach has some similarities to the one proposed in this paper as it also relies on validation data sets for hyperparameter tuning (Sec. VIII), our approach was designed in a way such that the configuration methods' (hyper) parameters are universally valid once determined.

III. Concept

The process of the CE is preceding the actual AD, as its output serves as the configuration of the AD tool, which takes raw log data and a configuration file as input. The log data is parsed into variables and the configuration dictates which detector should be employed, which variables each detector should analyze and some additional parameters are chosen (thresholds, window sizes, etc.).

The input of the CE is only the raw log data. The CE uses the same parser as the AD tool to extract variables and applies it on the whole dataset, yielding a table structure with rows corresponding to log lines (or instances) and columns corresponding to variables (or features), to enable data analysis. We then apply a set of methods on the data that assign variables to the appropriate detectors and define other parameters. This information is written into a configuration file that the AD tool then receives as input. Note, that variables are included when we speak of configuration parameters. The process of the CE consists of the following steps:

1) Parse raw log data (the input) into variables.
2) Select or filter variables based on their characteristics for each detector and compute other relevant parameters.
3) Optimize the initial parameter selection (see Sec. VI).
4) Pass information into a configuration file (the output).

IV. Detectors

The configuration methods of the CE were designed for the detectors implemented by the AMiner [3]. It employs a collection of different AD algorithms that use techniques such as text processing, time series analysis, association rule based approaches and more. The implementation of the AMiner[1] is used for the evaluation of this work.

For the application of the CE, four (out of 27) detectors, implemented by the AMiner, were chosen. These detectors were selected due to their diverse detection principles and they are the most used ones by the experts. Their AD principles determine which variables are suitable for them to operate effectively. A brief description of each detector is given below, including the type of input the detectors receive and the output they produce [3].

A. NewMatchPathValueDetector (NMPVD)

Anomalies are detected whenever a new and unknown value is found. Therefore, we want to train the detector with variables that form a limited set of values.

- **Input parameters**: individual variables.
- **Output**: anomalous events.

B. NewMatchPathValueComboDetector (NMPVCD)

This detector is basically an extension of the NMPVD. Anomalies are detected whenever a new and unknown value combination for the corresponding variable combination is found (within the same event). Therefore, one would want to pass variable combinations to the detector that form a limited set of value combinations. The number of possible variable combinations does not scale linearly with the number of available variables, it is therefore important to consider the computational cost of this detector's configuration process.

- **Input parameters**: variable combinations.
- **Output**: anomalous events.

C. EntropyDetector (ED)

The ED generates a frequency table for each character pair of each occurring value of a variable. After analyzing a large enough sample in the training phase, the relative frequencies of character occurrences should become stable. This allows for the detection of anomalies by summing the probabilities of all character pairs in a new value and flagging it as anomalous if the combined probability is below threshold ϕ. We call this probability critical value $P(x)$. We use symbol ϕ instead of θ to point out that this is a threshold which is directly passed to the AMiner as input and not a threshold used to adjust the configuration methods [3].

From the above we conclude that the behaviour of the critical values of variables in the training data is the important factor to consider.

- **Input parameters**: individual variables, a lower limit for critical values ϕ.
- **Output**: anomalous events, the critical values of the anomalous events.

D. ValueRangeDetector (VRD)

The VRD generates ranges for numeric values, detecting values outside these ranges and extending ranges when it is learning [3]. Variables with numeric values and limited ranges are therefore the right choice for this detector.

- **Input parameters**: individual variables.
- **Output**: anomalous events.

V. Configuration Methods

Since each detector requires different input parameters it has to be assessed individually for each method which parameter values are suitable. The first step in finding the right parameters is to assess which variables to choose. This configuration step can be automated by mapping the variable properties to the corresponding detection method and thereby classifying the variables into certain feature sets that suit the corresponding detector. The mapping process can be generalized for all detectors:

1) Choose detector.
2) Define data characteristic from detection method.
3) Expand characteristic to a measure of stability.

A simple example incorporating these steps is provided by the NewMatchPathValueDetector (1) of the AMiner which triggers an alert whenever a new and unknown value of a specified variable is found in a log line [3]. Consequently, we do not want to pass certain variables to this detector. Imagine a feature that has a different value in every occurrence (e.g. random variables — Sec. V-B). For this detector any learning would

[1]AMiner GitHub page https://github.com/ait-aecid/logdata-anomaly-miner; accessed 13-May-2024.

TABLE I
DEFINITIONS OF MATHEMATICAL EXPRESSIONS.

Expression	Description
$\mathcal{V}$	the set of all variables (whole dataset)
V	a set of variables; $V \subseteq \mathcal{V}$
x, y, z	variables of $\mathcal{V}$; $x, y, z \in \mathcal{V}$
x_i	the value of the i-th occurrence of x; $x_i \in x$
θ	thresholds used in configuration methods
$\lvert x\rvert : V \to \mathbb{Z}$	total count of occurrences of x

be irrelevant with this kind of values and it would trigger FP for every occurrence. The corresponding characteristic is therefore based on "unique occurrence" (3). The suitable measure of stability hence is the asymptotic decrease of new unique occurrences of variables over time (4). In other words, the occurrence of new and unknown values has to stabilize after some time or some number of events.

The following subsections describe the different configuration methods. To effectively use mathematical formulations in this context, a set of expressions is introduced in Table I.

A. Static Occurrence

Variables that have the same value in almost every sample in which they occur are classified as "static" [9] (e.g. $x = [A, A, \dots, A]$). In other words, a variable is static if the number of unique occurrences is equal to 1. The set of static variables is defined as:

$$V_{static} = \big\{x \in \mathcal{V} \mid |\{x_0, x_1, \dots, x_{|x|}\}| = 1\big\}. \tag{1}$$

where $|\cdot|$ denotes the cardinality of a set. Note, that a set only contains unique values by definition.

B. Random Occurrence

Variables are classified as "random" if there are unique values for most occurrences (e.g. $x = [A1, A2, A3, \dots]$). As it is possible that individual values of a variable x are occurring randomly, the occurrences of each value of x are counted by $Count(x_i) = \sum_{j=1} \delta(x_j, x_i)$ with δ as the Kronecker delta. If this number is below a certain threshold θ for one of the values of the variable, it is considered as random:

$$V_{random} = \big\{x \in \mathcal{V} \mid \exists x_i \in x : Count(x_i) < \theta\big\}. \tag{2}$$

It is meaningful to choose $\theta > 1$. The larger θ the less random the variable.

C. Stability

The stability of a variable depends on the considered characteristic. We call a variable "stable" regarding that characteristic if the corresponding curve approaches a constant value within the training period. Fig. 1 exemplarily shows the behavior of different values regarding the number of unique occurrences against the number of occurrences. One can see, the random variable has a new unique occurrence for every occurrence ($Count(x_i) = 1$) while the static one only has a single unique occurrence and is therefore constant. The line between them shows the behavior of a variable that could be classified as

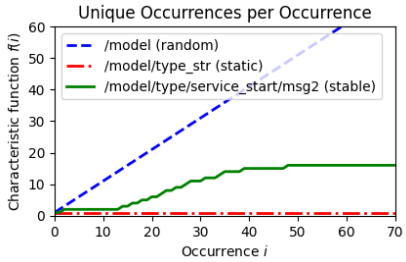

Fig. 1. Unique occurrences per occurrence of a static (red), stable (green) and random variable (blue).

stable as no new unique values occur at some point. Consequently, static variables are a subset of the stable variables. For many detectors there is some kind of stability involved since it implies some kind of learnable behavior for many detectors. The curves for static, stable and random occurrence can at most increase by 1 per occurrence. Fig. 1 therefore presents the entire spectrum of possible behavior.

To check whether a variable is "stable", a threshold curve is defined that acts as an upper limit for the curve $f(i)$, representing the data characteristic we are interested in. $i \in \mathbb{Z}$ is the number of occurrences of a variable x. This threshold curve is applied to the derivative $f'(i)$ representing the change in $f(i)$ per occurrence. For a stable variable, this curve should therefore approach 0 within the period of the training data. For the detectors covered in this work, we are actually not interested in the magnitude of change but whether a change has occurred or not. Consequently, $f'(i) \in \mathbb{R}$ should be an element of the binary space $\{0, 1\}$ and we define the boolean conversion (denoted by the subscripted b), which later allows us to define relative thresholds in the range $[0, 1]$:

$$f_b(i) = \begin{cases} 1 & \text{if } f(i) \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Thus, the function $f'_b(i)$ is 1 if a change in $f(i)$ occurred at occurrence i or 0 for no change.

Stability is based on the assessment of the mean values of the segments $s_m(i)$ with $m = 0, 1, ..., n_s - 1$. n_s being the number of segments. To be precise, the m-th segment of the function $f'_b(i)$ is:

$$s_m(i) = \begin{cases} f'_b(i) & \text{if } \frac{m}{n_s}|x| \leq i < \frac{m+1}{n_s}|x| \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The set of stable variables is then defined as:

$$V_{stable} = \left\{x \in \mathcal{V} \mid \frac{1}{|s_m|} \int_{\mathbb{R}} s_m(i)\, di \leq \theta_m \quad \forall m\right\} \tag{5}$$

$|s_m|$ is the length of each segment m. $|s_m|$ is not uniform if it is not divisible by n_s. Therefore, we define quotient $q = |x| : n_s$, remainder $r = |x| \mod n_s$ and:

$$|s_m| = \begin{cases} q + 1 & \text{if } m + 1 \leq r, \\ q & \text{if } m + 1 > r. \end{cases} \tag{6}$$

If each of the segment means $s_m(i)$ is below the thresholds θ_m, the corresponding variable is classified as "stable". The

thresholds θ_m represent a discrete threshold curve that serves as an upper boundary for the change in each segment of $f(i)$. $f'_b(i) \in \{0, 1\}$ represents this relation as the "relative change per segment". This is also convenient for the selection of the thresholds as we can define them within range $[0, 1]$.

For a stable variable $f'(i)$ should behave similar to an exponential decay, i.e., approach 0. It is therefore meaningful to say $\theta_m = e^{-cm}$ with a constant c determining the magnitude of decay. In general, any function with similar behavior to an exponential decay (e.g., $1/x$) could be used for θ_m.

The paragraphs above described stability generically, so that it can be applied to any function $f(i)$ representing the behavior of a certain data characteristic. The following describes the specific types of stability based on different data characteristics:

1) Stability by Unique Occurrence: The characteristic we are interested in is the behavior of the number of unique values over the training period as in Figure 1. Therefore, we take

$$f(i) = |\{x_0, x_1, x_2, \ldots, x_i\}| \tag{7}$$

and its (discrete) derivative

$$f'(i) = f(i) - f(i-1). \tag{8}$$

Hereby, we can assess whether the count of unique values of a variable is stable. For a variable classified as stable by occurrence, it can be assumed that it has a limited set of unique values. If the thresholds are reasonably set one can say that stability by occurrence is a weaker condition than static occurrence (Sec. V-A) but a stronger condition than non-random occurrence (Sec. V-B). Note, that the change of the count of unique values per occurrence can be at most 1 so that $f'(i) \in \{0, 1\}$ and therefore $f'(i) = f'_b(i)$.

2) Stability by Value Range: For the VRD one would want to pass variables with a limited range of numeric values. Consequently, we define a measure of stability based on the minimum-maximum range for variables containing only numeric values V_{num}. We have:

$$f_{min}(i) = \min(\{x_0, x_1, \ldots, x_i\}), \tag{9}$$
$$f_{max}(i) = \max(\{x_0, x_1, \ldots, x_i\}). \tag{10}$$

In order to fit the stability relation of Eq. 5, we take the derivatives and add both functions together to get:

$$f'(i) = |f'_{min}(i)| + |f'_{max}(i)|, \tag{11}$$

where $|\cdot|$ denotes the absolute value. Thus, this yields a function that is 0 when no change occurred in the minimum or maximum value of the variable and > 0 otherwise.

3) Example: For some data characteristic, e.g. "unique occurrence" (see Eq. 7), we have $\theta = [1, 0.5, 0.1]$ (previously chosen by e.g. hyperparameter tuning on a validation set, hence we know that these thresholds are a valid choice for the data characteristic at hand). For some variable x we observe a sequence of events $[a, b, a, c, a, a, d, a, a, a]$ (e.g. from log lines containing something like "type=a"), from which we subsequently count the number of unique occurrences per event, from which we obtain $f = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4]$. Consequently, we have $f' = f'_b = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0]$, thus $|x| = |f'| = 11$ and $n_s = 3$. Therefore, $q = 3$, $r = 2$ and $|s| = [4, 4, 3]$. The segments themselves are then $s = [[1, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0]]$ and their mean values $\overline{s} = [0.75, 0.25, 0.0]$. As $0.75 \leq 1$, $0.25 \leq 0.5$, $0.0 \leq 0.1$ are all true, we classify variable x as stable.

D. Co-occurrence

Two or more variables are co-occurring if they occur in the same event [6]. Some detectors require variable combinations as input such as the NMPVCD, which raises an alert whenever a new combination of values for a specified combination of variables is found. Since combinations do not scale linearly we have to take computational cost into account. The number of combinations is given by the binomial coefficient. For example, for audit log data it is easily possible to receive around 350 variables from the parser for all log lines of a dataset. Note, that a single audit log line contains far less variables (around 30). Also, this number strongly depends on the parser itself. For combinations of length 2 there are $61,075$ combinations or for length 3 there are already $7,145,775$. To reduce this number it is meaningful to filter certain variables beforehand based on their type — see Sec. VII.

The combinations are then selected by the assessment of co-occurrence. A combination c is selected if it occurs at least θ_{abs} times ("abs" for "absolute"). The set of valid combinations $C \subseteq \mathcal{P}_2(\mathcal{V})$ is therefore defined as:

$$C = \left\{c \in \mathcal{P}_2(\mathcal{V}) \mid |c| \geq \theta_{abs}\right\}, \tag{12}$$

with $|c|$ as the total count of occurrences of combination c and $\mathcal{P}_2(\mathcal{V})$ as the power set of $\mathcal{V}$ for combinations of 2 variables. We limit combinations to 2 variables to further decrease the computational effort for this step. However, a later step allows combinations of more than 2 variables.

The selection of θ_{abs} is not trivial since the number of co-occurrences can be rather arbitrary for different variable combinations and datasets. It is therefore meaningful to choose a relative threshold θ_{rel}. Since there are several variables in a combination that can occur with different frequencies, the threshold value is defined relative to the total occurrence of the most frequently occurring variable in a combination:

$$\theta_{abs} = \max\left(|c| \mid \forall v \in c\right) \cdot \theta_{rel}, \tag{13}$$

with v as an arbitrary variable in combination c.

Testing has shown that C often consists of too many combinations which can overwhelm the AD tool in terms of computational cost. Especially when running in online mode, the tool has to be efficient enough to process more events per time interval than events are occurring. To address this issue, we apply a simple graph theory method, because combinations can be represented as nodes connected by edges in a graph. For instance, the combinations (x, y), (x, z), (y, z), (v, y), (v, w) can be represented as the graph in Fig. 2.

Next, we merge all connected nodes. For the example at hand, this leads to the combination (v, w, x, y, z). This is

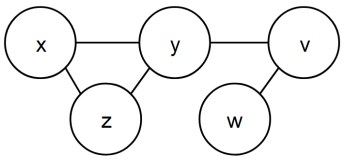

Fig. 2. Variable combinations represented as connected nodes in a graph.

possible as for the NMPVCD the set of 2-combinations and the merged combination are equivalent. For instance, assume the combinations (x, y), (x, z), (y, z) consisting of the variables x, y and z. For the NMPVCD these combinations are equivalent to the combination (x, y, z). Since we want to reduce the total number of combinations, the latter is preferred.

E. Character Pair Probabilities

The "character pair probability" is the probability of occurrence of a character pair in a variables' value. Character pairs are consecutive fragments of a character sequence.

This approach is closely tied to the one of the ED (see Sec. IV-C) as we utilize the same critical value $P(x)$ computed fro each value of a variable. Further, we compute the mean of all critical values as $\overline{P}(x) = \sum_i P(x_i)/|x_i|$ for each variable x, resulting in an indicator for how likely the average value of x is to occur. Variables with a mean critical value below a certain threshold θ_{CPP} are considered too unpredictable as their character pair probabilities (CPP) are too arbitrary or the training samples were not enough for them to stabilize. The other variables are selected as input for the detector:

$$V_{CPP} = \left\{ x \in \mathcal{V} \mid \overline{P}(x) \geq \theta_{CPP} \right\}. \tag{14}$$

Threshold θ_{CPP} can be understood as the "minimum mean critical value" and has to be chosen through empiric validation. It has been observed that a rather high value, between 50% and 80%, yields better results than lower values.

In case of the ED, its configuration expects a specific threshold parameter ϕ that decides if a critical value of an instance is anomalous. $\phi \in [0, 1]$ is an indicator for how unlikely a value has to be in order to be detected as an anomaly. We compute it by taking the minimum of all critical values $P(x_i)$ of selected variable x. From this minimum we subtract a certain offset δ (in practice, some value between 0.01 to 0.1 is appropriate) to have a buffer between the least likely values that were still considered as normal behavior, so that the same values or the ones with similar critical values are not considered as anomalies after training:

$$\phi(x_i) = \min_i P(x_i) - \delta. \tag{15}$$

As we take the minimum of all critical values as ϕ, it will be strongly affected by potential outliers with very low critical values within the training data. However, since we consider this as normal behavior, this trade-off is acceptable. Also, this corrects the potentially poor selection of a certain variable by lowering ϕ to a level where only very unlikely values will trigger an alert. In practice, it is meaningful to limit ϕ to a certain range since it is possible that the minimum critical value for a certain variable can be arbitrarily low or high within the range of $[0, 1]$.

VI. Parameter Optimization

In the following, we investigate an approach to systematically optimize a configuration by the output obtained from the AD tool running on the training data. The objective is to identify false positive sources in the configuration and to adapt or remove them in order to minimize the occurrence of FPs. These sources can either be variables, options and thresholds, or other numeric parameters listed in the configuration of a detector. This optimization approach is therefore a measure to reduce the sensitivity of the detector to the training data and thereby also the likeliness of overfitting. An AD tool running on anomaly-free data cannot produce true positives (TP). It is therefore not possible to solve this as a minimization problem.

Each of the detectors requires a set of variables, which can be obtained by the methods of Sec. V or by plugging in an already existing configuration. To know how the initial configuration performs regarding the amount of FP, the detection tool has to be run with the given data. For the sake of generalization, a cross-validation technique is used here. This method splits the data K times based on the number of events, while maintaining the order of, first, the train and secondly, the test split. It is also possible to split by time interval but testing has shown better results when splitting by sample size.

Goldstein and Uchida name two forms of output an AD tool can have: labels and scores [25]. However, some of the AMiner's detectors include additional information in their alerts (for explanatory reasons) which we make use of. From the alert report of the AD tool, we can extract the number of FP, which event caused the alert (a timestamp or line index) and optionally information of what caused each alert (e.g. the critical value calculated in the ED — Sec. IV).

Note, that the detected anomalies are considered as point anomalies. The number of FP and timestamp t_n of event n can be used to determine if an action is necessary for the corresponding detector instance d. Two conditions are constructed that assess if an action has to be taken:

1) Check if FP is greater than a certain threshold θ_1 for each detector instance. Since the tool is run K times, we have an array of FP_k for each run k. The data splits are usually not equally sized. Therefore, we take a weighted mean [26], denoted by the bar with the subscripted w, where the weights increase linearly with the size of the corresponding split:

$$\overline{(FP)}_w = \Big(\sum_{k=1}^{K} \frac{k}{K} \cdot FP_k\Big) / \Big(\sum_{k=1}^{K} \frac{k}{K}\Big). \tag{16}$$

   The condition is then defined as $\overline{(FP)}_w > \theta_1$.
2) Similarly, we limit the amount of FP per time interval $\Delta t = t_n - t_0$ with t_0 and t_n as the timestamp of the first and last alert from the corresponding detector instance. Therefore, $\overline{(FP/\Delta t)}_w > \theta_2$.

Since there are several detectors and different ways to adapt their different parameters and possible combinations, the scope of this procedure was limited to the adaptation of numeric thresholds — see Sec. VI-A.

If the corresponding detector expects an input parameter in the form of a threshold, the value of the threshold is adjusted so that conditions $\overline{(FP)}_w > \theta_1$ and $\overline{(FP/\Delta t)}_w > \theta_2$ are fulfilled. If the value cannot be adjusted to comply with the conditions, the corresponding detector instance will be pruned from the configuration. If the corresponding detector does not expect a threshold as input parameter (e.g. only variables), they will also be pruned from the configuration. If the optimization approach removes every instance from a configuration the optimized configuration is discarded and the original one is returned.

A. Threshold optimization

The information of what caused an alert can be used to adapt the thresholds of detector instances, given that they expect one (or more) as input parameter. To be precise, we expect that there is a critical value ϕ^* listed in the alert report of the detection tool that was either too high or too low compared to the threshold that was specified in the corresponding detector instance, so that an alarm was triggered.

To adapt the given threshold of a detector instance, the critical value for each alarm, given that there is one, is extracted from the alert report. This is done for each of the K runs of the detection tool. We yield multiple critical values for a detector instance for each run k. Out of all of these, the minimum is taken. The new adapted threshold of the detector instance is therefore:

$$\phi_i^{new} = \min_k(\min_n \phi^*_{i,k,n}) - \delta, \tag{17}$$

with n as the number of critical values in a single run and offset δ which has the same functioning as in Sec. V-E.

Even tough this procedure is applicable to a wider range of different AD algorithms, in this paper it is only relevant for the ED, as this is the only detector of the selection that returns such a critical value for its detected anomalies.

VII. Meta-Configuration

This section describes how the configuration for each detector is assembled from the methods described in Sec. V. In case of the selected detectors, the mappings require two types of operations. For each detector we can

- **filter variables** from the set of given variables V. Only the remaining variables V' are then passed on to the subsequent operation.
- **select parameters** such as variables (or combinations) from V' (or power set $\mathcal{P}(V')$) or thresholds, window sizes, options, etc. that are passed to the detector.

Based on these operations we can define the mappings for the detectors. Note, that the order of the operations represents the order in which they are applied. The input for the first operation is the set of all variables $\mathcal{V}$, from which we can filter variables that are unnecessary or lead to an unfeasible computational effort for certain methods before they are passed to that method. In the following, we list which configuration methods are applied to which detector. Note, that the parameters named below are the hyperparameters of the configuration methods of the CE from Sec. V. Their concrete values were determined via hyperparameter tuning on the validation sets.

- **NewMatchPathValueDetector (NMPVD)**:
  - **Filter static variables** as they blow up the configuration unnecessarily and are trivial to detect.
  - **select stable variables (by occurrence)** because they contain a limited set of values. Thereby, we use an exponential decay for $\theta_m = e^{-cm}$ with $m = 0, 1, \ldots, 4$, $c = 1.8$ (Sec. V-C).
- **NewMatchPathValueComboDetector (NMPVCD)**:
  - **Filter random variables** as they lead to random combinations which do not form a limited set of value combinations ($\theta = 2$; see Sec. V-B).
  - **Filter static variables** as they form trivial combinations (Sec. V-A).
  - **Filter variables by minimum occurrence** to make sure that only variables with significantly many occurrences are combined. Hereby, we determine the minimum amount of occurrences 0.5% of the total number of instances of the dataset.
  - **Select variable combinations based on co-occurrence** as only combinations of variables are relevant that co-occur at least a certain amount of times ($\theta_{rel} = 0.1$; see Sec. V-D).
- **EntropyDetector (ED)**:
  - **Filter static variables** as their critical values are also static. A change in a static variable is trivially detectable by less complex detectors.
  - **Select parameters by character pair probabilities** as the values of variables are on average rather likely to occur (if θ_{CPP} is reasonably set), and thus represent a pattern from which the ED is effectively able to learn a normal behavior ($\theta_{CPP} = 0.7$, $\phi \in [0.0, 0.9]$, $\delta = -0.05$; see Sec. V-E).
- **ValueRangeDetector (VRD)**:
  - **Select stable variables (by value range)** as they are numeric and their minimum-maximum ranges are limited. We use $\theta_m = e^{-cm}$ with $m = 0, 1, \ldots, 4$, $c = 1.8$ (Sec. V-C).

VIII. Evaluation

All results were produced with the same evaluation environment. The system runs 64-bit Windows 10 with Ubuntu 20.04 via Windows Subsystem for Linux (WSL) and uses an Intel Core i7-8665U CPU with a base clock speed of 1.90 GHz and 16GB RAM. The implementation of the CE and the expert's configurations are available on GitHub[2].

[2]Configuration-Engine GitHub page https://github.com/ait-aecid/aminer-configuration-engine; accessed 13-November-2024.

TABLE II
DATASET STATISTICS. "V" OR "T" IN "V/T" INDICATES WHETHER THE DATASET WAS USED FOR VALIDATION (V) OR TESTING (T).

Name	Training samples	Test samples	Point anomalies	Collective anomalies	V/T
Apache Access datasets					
russellmitchell	2884	8300	7696	7	V
fox	9058	413948	410841	16	V
harrison	19604	420790	415376	352	V
mail.onion.com	53004	28959	6429	19	V
shaw	8050	7696	5226	6	T
santos	6752	9032	7794	7	T
wardbeck	32454	9647	5299	9	T
wheeler	7848	433072	431560	53	T
wilson	25130	438743	428116	89	T
mail.spiral.com	65811	34634	7370	30	T
mail.insect.com	118549	50791	6973	30	T
mail.cup.com	115443	33091	6789	28	T
Audit datasets					
russellmitchell	1859	457	9	2	V
fox	2078	809	19	3	V
harrison	2454	376	24	3	V
shaw	2608	787	19	3	T
santos	1968	295	19	3	T
wardbeck	2645	274	19	3	T
wheeler	2693	148	14	3	T
wilson	2622	851	22	3	T

Validation and testing was conducted with the Apache Access and audit datasets given in Tab. II. The datasets that start with "mail" are from AIT Log Data Set V1.0 [20] while the rest origins from AIT Log Data Set V2.0 [21]. Each dataset was divided into a training and a test or validation set. The training set consists of all instances up to (but excluding) the first attack and the test set from the first attack to the last entry. There are no attacks in the training set.

Goldstein and Uchida [25] distinguish three types of anomalies: point, collective and contextual anomalies. For our case, we define a point anomaly as a single log line and a collective anomaly, or attack period, as a sequence of log lines that belong to a specific attack type and are subsequently occurring with no non-attack log lines in between. Attack periods separated by non-attack lines are treated as different collective anomalies or attacks. An anomaly is considered as detected if at least one log line of the anomaly is detected. In this paper we do not consider contextual anomalies.

In order to demonstrate the effectiveness of each detector for both the Configuration-Engine (CE) and the baseline, we assess the performance with respect to precision, recall and F1-score of each detector individually. The graphs in Fig. 3 and Fig. 4 show the averaged detection performance of each detector configured with the CE compared to the performance of manually created configurations from three different experts in the domain of cyber security. Each of which has deep knowledge about the AMiner and is involved in its development, thus setting a high level baseline performance of the manually created configurations. The evaluation metrics shown in the graphs are computed as the mean of the configurations' performances on the datasets, with and without the optimization. For the optimization, we fixed the number of allowed FP to $\theta_1 = 10$ and the number of FP per minute to $\theta_2 = 0.05$ (both weighted). The number of splits was set to $K = 3$ (see Sec. VI). One can see that the performance of the configurations

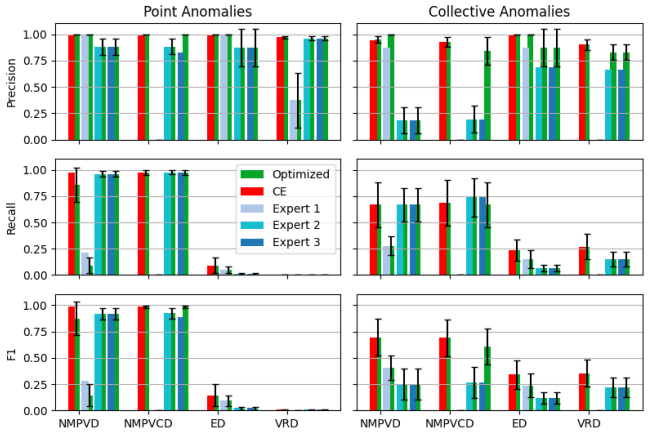

Fig. 3. Performance of each detector for Apache datasets with and without optimization.

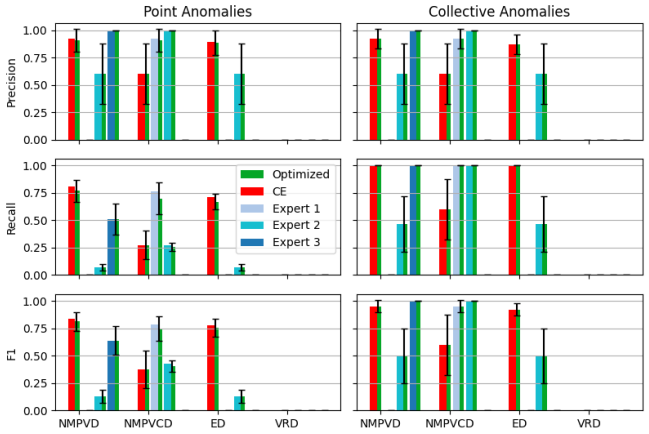

Fig. 4. Averaged performance of each detector for audit datasets with and without optimization.

generated by the CE competes with the ones of experts' configurations. Note, that no numeric variables are given in audit data, thus the VRD is not applicable there. Especially for Apache data the optimization exhibits great potential for increasing the precision at almost no reduction in recall. An increase in precision is especially favoured over a high recall score if a collection of multiple detectors is employed for simultaneous AD, as different detection algorithms typically detect different anomalies.

For audit data the optimization seems less effective, but does also not decrease the performance. The lower performance on audit data is probably caused by the less structured nature of audit logs. Typically audit log data contains more diverse events than Apache log data, which makes it more difficult to apply AD methods effectively. However, some of the experts' configurations were able to configure specific detectors more effectively. This means that there are still possible improvements for the CE, potentially in the form of stricter parameter selection methods or stricter thresholds.

Fig. 5 shows the runtimes of the CE for generating the configurations for the detectors individually averaged over each dataset. One can see that the runtimes are mostly relatively low, whereas the configuration method for the NMPVCD

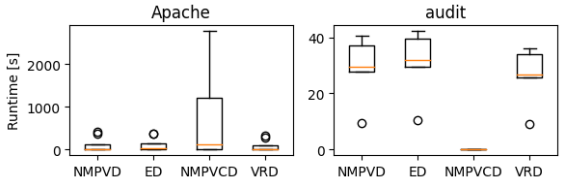

Fig. 5. Averaged runtimes for each detector for Apache and audit datasets.

exhibits larger execution times for Apache data, because of the strong scaling of the method and the large number of samples in some of the Apache datasets.

A. Similarity of Configurations

The previous evaluation approach is solely based on the evaluation of traditional machine learning metrics, but the generated configurations themselves might hold additional information worth investigating. Therefore, assess the similarity of automatically generated configurations to the experts' manually created ones, and to each other. The configurations are given as associative arrays, consisting of different keys and value types. Thus, it is not straightforward to measure their similarity. A general approach is to assess the similarity of all present detector-variable pairs as this defines what features of the data the detector investigates. In case of the experts' configurations, it shows which features they consider important.

A detector-variable pair is given as (d_a, A) with detector d and the set of variables A for the configuration a which is defined as a set of detector-variable pairs:

$$a = \{(d_a, A)_i \mid i = 1, ..., |a|\}, \tag{18}$$

where $|\cdot|$ denotes the cardinality of a set. For the ED and the VRD we have $|A| = 1$ since they take a single variable as input, but the NMPVCD expects combinations of variables, which is why we define A as a set of variables and $|A| \geq 1$. The comparison of two configurations a and b is then between the sets of variables A and B for which $d_a = d_b$ holds. For this comparison we use the Jaccard similarity coefficient $J(A, B) = |A \cap B|/|A \cup B|$ [27]. The comparison of pairs that are not from the NMPVCD can only yield $J(A, B) = 0$ or $J(A, B) = 1$. If it would not be for the NMPVCD we would not need the Jaccard index. We define:

$$\widetilde{J}(a, b) = \begin{cases} J(A, B) & \text{if } d_a = d_b, \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

to impose the condition. The similarity $s_{ab} \in [0, 1]$ of configuration a and b is then the sum over all Jaccard indices where $d_a = d_b$ divided by the size of the configuration b:

$$s_{ab} = \frac{1}{|b|} \sum_{i=1}^{|a|} \sum_{j=1}^{|b|} \widetilde{J}(a_i, b_j). \tag{20}$$

The sums over the modified Jaccard indices $\widetilde{J}$ are equivalent to the sum of every element of a matrix where only the elements are non-zero where $d_a = d_b$.

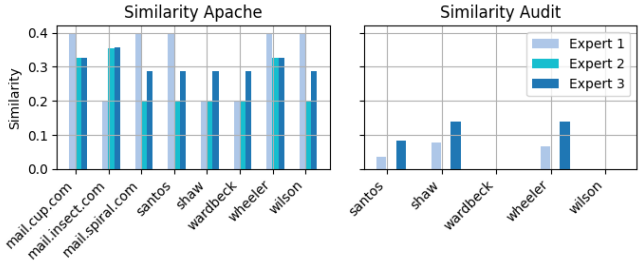

Fig. 6. Similarity between the CE and the expert configurations for different Apache and audit datasets.

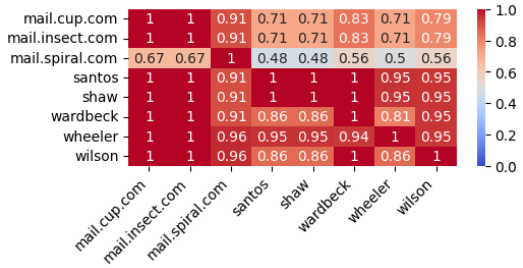

Fig. 7. Heatmap of similarities across the configurations generated on different Apache datasets.

For the graphs in Fig. 6 we use a as the automatically generated configuration of the CE and b as the expert's configuration. Since the formula divides by the size of the expert configuration b the similarity can be understood as: the percentage of detector-variable pairs in the expert configuration b that are also given in configuration a.

Fig. 3, Fig. 4, and Fig. 6 reveal that high performance can be achieved by significantly dissimilar configurations. For audit the similarity between the CE's and the experts' configurations is almost 0, while their performance strongly differentiates for each. Meanwhile, the CE's configurations for the audit datasets are highly dissimilar compared to each other (Fig. 8). This is most likely caused by the large number of variables the parser identifies in the audit log lines.

The heatmaps from Fig. 7 and Fig. 8 show how similar the configurations from the CE are with each other compared across different datasets. Note, that the similarities for dataset santos are 0, because no suitable detector instances could be generated for it (configuration is empty). As we also get a satisfying performance for precision and recall across almost all datasets, it follows that the Apache configurations of the CE are effectively portable from one dataset to another, due to their high similarity. This further implies that the important variables for the detection are mostly the same across the different Apache datasets (of the same type). At least for the selection of detectors and datasets, this suggests that it is possible to define a single suitable configuration for all Apache datasets — at least in terms of the variables. On the other hand, the low similarities between audit configurations show that a specific configuration is necessary for each audit dataset. This highlights the importance of an automated configuration process.

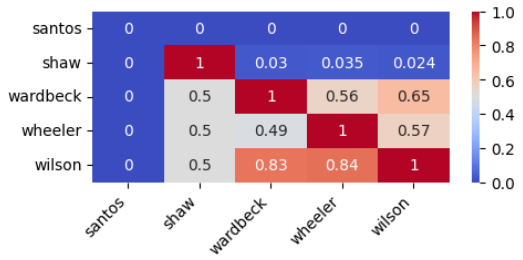

Fig. 8. Heatmap of similarities across the configurations generated on different audit datasets.

IX. Limitations and Future Work

One area of interest for future work is the expansion of the range of configuration methods for other detectors. The existing framework is robust, yet only a small selection of configuration methods is defined (covering 7 out of 27 detectors of [3]) and the defined methods leave potential for refinement. While applying the CE directly to other AD methods is difficult, since a configuration method has to be defined specifically for distinct methods, it is possible to apply its concept to every algorithm where a data-driven parameter selection is required.

As mentioned previously, the performance for audit is worse than for Apache data, especially for the NMPVCD. Future work should investigate the impact of different log data types or non-standardized datasets on the performance more closely. This will help finding a better universal parameter setting for the CE across different data types or point out differences in the requirements to the CE's settings for different data types. In general, the less structured and stable the logs are, the more difficult it is to extract the important variables. For our future work, we also plan to additionally create a detailed benchmark against other recent AD solutions to better integrate the CE into the broader research field.

The CE is limited to offline processing. In order to integrate the CE into an online AD framework, to adapt AD algorithms to changes in the data over time, it could be applied on batches of log lines. It is also limited to the existing parsers of the AMiner, which are basically predefined log templates for a set of different data types. Incorporating better parsing into the AMiner framework would also increase the CE's applicability to a wider range of data types and might also improve the performance, since the downstream tasks are inherently depending on log parsing [28].

X. Conclusion

This work introduces the Configuration-Engine (CE), a novel approach to automate the configuration of AD algorithms in a semi-supervised manner. The CE analyzes system log data under the assumption of the data being anomaly-free. The core of the CE is the classification of variables into sets based on their character and behavior over time. The objective is the automation of the tedious configuration process that usually requires domain expertise. The configuration methods based on the characteristics stability, co-occurrence, and character pair probabilities serve as an extension to their associated detection algorithms, by transforming the extracted information from the data into the input parameters of the associated detectors.

The approach is demonstrated and evaluated with the AMiner and its detectors, yet the general approach of defining a configuration method for a detector is applicable to any kind of AD algorithm. On the other hand, the defined configuration methods are specifically tailored to the specific detectors of the AMiner or to those utilizing similar detection techniques. With the CE, the AMiner detectors proved to achieve satisfying performance competing with the performance of expert crafted configurations. Furthermore, the optimization approach was able to effectively improve the precision in most cases, with almost no reduction in recall.

Moreover, it is shown that experts' configurations are considerably dissimilar to the automatically generated configurations of the CE. At the same time, the CE's configurations are highly similar to each other across different datasets for Apache data, indicating the possibility of effective portability of configurations across different Apache datasets, while the dissimilarity between audit datasets highlights the importance of automated configuration.

Acknowledgement

Funded by the European Union under GA no. 101121403 (NEWSROOM) and GA no. 101103385 (AInception). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them. Co-funded by the Austrian FFG Kiras project ASOC (905301).

References

[1] I. Lella, C. Ciobanu, E. Tsekmezoglou, M. Theocharidou, E. Magonara, A. Malatras, R. Svetozarov Naydenov, *et al.*, "Enisa threat landscape 2023: July 2022 to June 2023," 2023.

[2] "Crowdstrike 2024 global threat report," 2024. Retrieved from https://www.crowdstrike.com/resources/reports/crowdstrike-2024-global-threat-report/, accessed 24-June-2024.

[3] M. Landauer, M. Wurzenberger, F. Skopik, W. Hotwagner, and G. Höld, "Aminer: A modular log data analysis pipeline for anomaly-based intrusion detection," *Digital Threats*, vol. 4, mar 2023.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing surveys*, vol. 41, no. 3, pp. 1–58, 2009.

[5] A. Taha and A. Hadi, "Anomaly detection methods for categorical data: A review," *ACM Computing Surveys*, vol. 52, pp. 1–35, 05 2019.

[6] M. Landauer, G. Höld, M. Wurzenberger, F. Skopik, and A. Rauber, "Iterative selection of categorical variables for log data anomaly detection," in *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I 26*, pp. 757–777, Springer, 2021.

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

[8] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations Management (IPOM 2003) (IEEE Cat. No.03EX764)*, pp. 119–126, IEEE, 2003.

[9] M. Wurzenberger, G. Höld, M. Landauer, and F. Skopik, "Analysis of statistical properties of variables in log data for advanced anomaly detection in cyber security," *Computers Security*, vol. 137, p. 103631, 2024.

[10] M. Kloft, U. Brefeld, P. Düessel, C. Gehl, and P. Laskov, "Automatic feature selection for anomaly detection," in *Proceedings of the 1st ACM Workshop on Workshop on AISec*, AISec '08, (New York, NY, USA), p. 71–76, Association for Computing Machinery, 2008.

[11] D. M. Tax and R. P. Duin, "Support vector data description," *Machine Learning*, vol. 54, pp. 45–66, 2004.

[12] C. Pascoal, M. R. de Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco, "Robust feature selection and robust PCA for internet traffic anomaly detection," in *2012 Proceedings IEEE Infocom*, pp. 1755–1763, 2012.

[13] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), p. 1285–1298, Association for Computing Machinery, 2017.

[14] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.," in *IJCAI*, vol. 19, pp. 4739–4745, 2019.

[15] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, (New York, NY, USA), p. 807–817, Association for Computing Machinery, 2019.

[16] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log Anomaly Detection via BERT," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2021. ISSN: 2161-4407.

[17] J. Qi, S. Huang, Z. Luan, S. Yang, C. Fung, H. Yang, D. Qian, J. Shang, Z. Xiao, and Z. Wu, "LogGPT: Exploring ChatGPT for Log-Based Anomaly Detection," in *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 273–280, Dec. 2023.

[18] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, (New York, NY, USA), p. 117–132, Association for Computing Machinery, 2009.

[19] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," *CoRR*, vol. abs/2008.06448, 2020.

[20] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, and A. Rauber, "Have it your way: Generating customized log datasets with a model-driven simulation testbed," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 402–415, 2021.

[21] M. Landauer, F. Skopik, M. Frank, W. Hotwagner, M. Wurzenberger, and A. Rauber, "Maintainable log datasets for evaluation of intrusion detection systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 3466–3482, 2023.

[22] M. Wurzenberger, F. Skopik, M. Landauer, P. Greitbauer, R. Fiedler, and W. Kastner, "Incremental clustering for semi-supervised anomaly detection applied on log data," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[23] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40, 2017.

[24] O. Anser, J. François, and I. Chrisment, "Automated machine learning configuration to learn intrusion detectors on attack-free datasets," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, pp. 1–7, 2024.

[25] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLOS ONE*, vol. 11, pp. 1–31, 04 2016.

[26] W. G. Cochran, *Sampling Techniques*. John Wiley & Sons, 1977.

[27] L. da F. Costa, "Further generalizations of the jaccard index," *CoRR*, vol. abs/2110.09619, 2021.

[28] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, (New York, NY, USA), pp. 1095–1106, Association for Computing Machinery, July 2022.