Cyber Intrusion detection in Industry 4.0 Data using Machine Learning

Md Ismail Hossain[1], Kakoli Khatun[2], Imtiaz Ahmed[3]
New Mexico Institute of Mining and Technology, USA[123]
Email: [1] statistician71@gmail.com, [2] kakoli.khatun@student.nmt.edu, [3] imtiaz.ahmed@student.nmt.edu

***Abstract*—Industry 4.0, also known as the Fourth Industrial Revolution, is characterized by the incorporation of advanced manufacturing technologies such as the Internet of Things (IoT), Artificial Intelligence (AI), and automation. With the increasing adoption of Industry 4.0 technologies, it becomes crucial to implement effective security measures to safeguard these systems from cyber attacks. The development of intrusion detection systems (IDS) that can detect and respond to cyber threats in real-time is crucial for securing Industry 4.0 systems. This research topic seeks to investigate the various techniques and methodologies employed in developing IDS for Industry 4.0 systems, with a particular concentration on identifying the most effective solutions for protecting these systems from cyber attacks. In this study, we compared supervised and unsupervised intrusion detection algorithms. We utilized data collected from heterogeneous sources, including Telemetry datasets of IoT and The industrial Internet of things (IIoT) sensors, Operating systems (OS) datasets of Windows 7 and 10, as well as Ubuntu 14 and 18 TLS and Network traffic datasets simulated by the School of Engineering and Information Technology (SEIT), UNSW Canberra @ the Australian Defence Force Academy (ADFA). The preliminary results of IDS accuracy are extremely encouraging on the selected data for this study (Windows OS and Ubuntu OS), which motivates the continuance of this line of inquiry using a variety of other data sources to formulate a general recommendation of IDS for Industry 4.0.**

***Index Terms*—Cybersecurity; Intrusion detection;Machine Learning (ML); Supervised ML; Unsupervised ML; Industry 4.0.**

I. Introduction

In 2011, the fourth industrial revolution dawned upon us with the introduction of the "Industry 4.0" concept by a German government initiative. This revolution signifies the deep integration of transformative technologies, such as artificial intelligence, robotics, and the Internet of Things (IoT), into modern manufacturing paradigms [1]. As this shift gathers momentum globally, it heralds unparalleled efficiencies and innovation. Yet, it simultaneously unveils intricate cybersecurity challenges, primarily arising from the melding of physical, digital, and biological domains [2].

Central to Industry 4.0 is the IoT, an extensive network of interconnected devices operating in synchrony. Its omnipresence, coupled with a vast device ecosystem, unfortunately, makes it a prime target for cyber adversaries [3]. These malevolent entities exploit vulnerabilities within the IoT, leading to an array of threats, from unauthorized data breaches to major operational disruptions. The repercussions of such breaches can be severe, threatening production, financial stability, and even human safety.

To address these escalating threats, Intrusion Detection Systems (IDS) have been pivotal. Traditional IDS primarily rely on known attack signatures, providing a line of defense based on previously identified threats. However, the ever-evolving landscape of cyber threats demands a more adaptive and proactive approach. This need has given rise to Machine Learning-based Intrusion Detection Systems (ML-based IDS) [4]. Unlike their traditional counterparts, ML-based IDS leverage machine learning to evolve with changing network patterns, enabling them to identify both established and emerging threats efficiently. Their prowess in handling vast datasets and discerning nuanced threat indicators makes them indispensable in safeguarding the multifaceted infrastructures of Industry 4.0 [5], [6],[7].

However, the efficacy of ML-based IDS is deeply rooted in the caliber of their training data. These systems flourish when trained on comprehensive datasets that encompass diverse network behaviors. Yet, sourcing such datasets, especially in niche sectors like the Industrial Internet of Things (IIoT), poses challenges, often attributed to data scarcity or inherent imbalances [8].

In summation, as the globe increasingly aligns with the Industry 4.0 paradigm, the urgency to bolster its foundational elements, especially the IoT, escalates. The imperative for advanced IDS that can adeptly navigate contemporary cyber threats becomes paramount. The subsequent sections of this paper embark on a deeper exploration of this domain: Section II offers a literature review, Section III details our datasets and research methodologies, Section IV unveils our experimental insights, and Section V summarizes our core findings.

II. Literature Review

The rise of Industry 4.0 and the proliferation of IoT devices have necessitated robust intrusion detection systems (IDS) to safeguard computer networks. Machine learning (ML) has emerged as a pivotal tool in this endeavor, with its adaptability allowing it to identify and counter evolving cyber threats [9].

Over the years, a multitude of research has been carried out using well-established datasets like KDDCUP99, NSL-KDD [10], UNSW-NB15 [11], CICID2017 [12], and the recent addition of ToN IoT [13]–[19]. The latter, being the most recent and diverse, encompasses multiple OS through

several virtual machines and reflects a more realistic network environment.

Anomaly detection, a core component of IDS, seeks to flag deviations from standard patterns, indicating potential malicious activities. ML-based anomaly detection models these patterns using statistical measures of system features, providing an edge in identifying previously unknown threats [20]. Techniques ranging from neural networks to support vector machines have been developed as classifiers to discern whether incoming network traffic is benign or malicious [9].

Supervised anomaly detection remains at the forefront of this domain. By training on labeled datasets, it captures typical behavior patterns, facilitating the classification of new instances as normal or anomalous. This approach has been extensively explored using algorithms like k-nearest neighbor (KNN), neural networks (NN), and support vector machines (SVM). For instance, using the NSL-KDD dataset, the K-NN algorithm was found to be a top performer with an accuracy of 99.403% over Decision trees (DT), Naïve Bayes (NB), Artificial neural network (ANN), k-NN algorithm, and Support vector machines (SVM) [21]. However, despite their robust performance on labeled datasets [22] [23], supervised techniques can face challenges when confronted with novel anomalies.

Contrastingly, unsupervised anomaly detection zeroes in on inherent data patterns without the need for labeled training. While methods like point anomaly detection have shown proficiency in detecting novel anomalies [24], they sometimes yield false positives or negatives. Studies such as Lazarevic et al. [25] have provided valuable insights into the comparative performance of supervised and unsupervised techniques.

The ToN IoT datasets, being the latest in this lineage, have served as an invaluable resource for researchers. For instance, in a study [26], seven supervised models and a deep learning method were employed to understand the impact of dataset heterogeneity on detection performance. The findings emphasized the prowess of RF and CART in intrusion detection across varied sensor data. Another study [27] delved into the predictive power of distinct feature sets for attack detection across datasets, including ToN IoT. Further, research [27] utilizing ten different learning methods on the ToN IoT dataset underscored the superior accuracy of the CatBoost algorithm.

Despite the strides made in the field, challenges remain. For example, the optimal application of supervised and unsupervised algorithms to specific ToN IoT data (Windows 7, 10, and Linux) remains an area of exploration. This research aims to address such challenges, drawing inspiration from prior works [26][28][29].

In conclusion, the literature underscores the intricate landscape of ML algorithms for intrusion detection, enriched by systematic reviews and comparative studies. The continuous evolution of cyber threats mandates ongoing research to refine and perfect IDS mechanisms.

III. The Significance of the ToN IoT Dataset

The choice of dataset is paramount in intrusion detection research, as it greatly influences the study's applicability and relevance. Among the datasets available, the ToN IoT dataset emerges as a superior choice for several compelling reasons:

1) Comprehensiveness and Diversity: The ToN IoT dataset offers a holistic view, encompassing a myriad of data sources, including telemetry data from IoT/IIoT services, operating system logs, and network traffic [30]. This dataset stands out with its heterogeneous data sources, addressing a gap found in other datasets [30]. It further enhances its relevance with a vast collection of over 5 million samples, amalgamating both malicious and benign data [31].
2) Rich Feature Set: A limitation evident in prior datasets is the paucity of features, especially in IoT-based IDS datasets [32]. The ToN IoT dataset bridges this gap with a comprehensive set of features, making it an optimal choice for evaluating AI-driven security applications [33]. Its focus on daily home usage devices provides a fresh perspective, differing from datasets that lean heavily towards academic network traffic [32].
3) Scalability and Heterogeneity: Many existing datasets falter when it comes to scalability and heterogeneity ([34]. However, the ToN IoT dataset shines in this aspect by integrating distributed data sources from varied IoT services, operating systems, and network traffic [33]. This diversity is essential for encapsulating the multifaceted nature of IoT network intrusion data, thereby augmenting the efficacy of intrusion detection systems [34].
4) Real-World Applicability: The ToN IoT dataset mirrors a medium-scale network, enhancing its real-world applicability ([30]. Its data, derived from a testbed architecture, encompasses detailed audit traces from Linux operating systems, including hard disk, memory, and process logs [33]. Such granularity in data strengthens the dataset's utility for appraising intrusion detection mechanisms in IoT settings [31].

In summation, the ToN IoT dataset surpasses its predecessors by addressing their limitations and offering a plethora of data sources, an enriched feature set, unmatched scalability, and genuine real-world representations. Its attributes underscore its significance, making it an indispensable tool for scholars and professionals navigating the realms of intrusion detection and IoT security

IV. Our Approach

A. *High-level Overview*

As we have mentioned, the industry 4.0 data provided by UNSW Canberra @ the ADFA [13]–[19] is a very robust data set to carry out different ML algorithms and testing their performance. Lots of active research is still ongoing and we have not found any unique research in fitting supervised and

unsupervised models done on Windows 7 & 10 and Linux-based OS data (Windows/Linux dataset-activities of desk, process, processor, memory, and network activities) in intrusion detection. Inspiring from this point of view, we have selected two supervised models (DT, and SVM) and two unsupervised models (SOM and K-means clustering). The most important reason for selecting these unsupervised models is that they can handle completely new types of attacks, although their performance is not as good as supervised models, as we have seen in [28]. Our research objective is to classify and cluster the data into two categories: "Attacked" and "Normal".

B. *Testbed Details*

The testbed for the ToN_IoT data that was developed by the University of New South Wales (UNSW) in Canberra is a real-world and massively scalable network that was built at the IoT Lab of UNSW Canberra Cyber. This testbed links multiple physical systems, virtual machines, hacking platforms, cloud platforms, fog platforms, and IoT sensors in order to simulate the scalable and composite nature of IIoT environments [35]. The testbed components are mainly organized into three layers: Edge, Fog, and Cloud. The Edge layer consists of IoT sensors and physical network components, the Fog layer includes virtual machines and hacking platforms, the Cloud layer encompasses cloud platforms [36].

This testbed was used to create the ToN_IoT Telemetry dataset, which contains data from heterogeneous sources gathered from IoT and IIoT sensors, OS data, and network traffic datasets [35]. We can show the layers by the following figure as used in [26]:

C. *Dataset Overview*

According to the study, the ToN IoT datasets [26] include heterogeneous data sources from IoT/IIoT service telemetry data, OS logs, and network traffic from a realistic representation of a medium-scale network designed at the Cyber Range and IoT Labs at UNSW Canberra. The primary focus of this study is the proposed dataset of IoT/IIoT service Telemetry data and their characteristics. The ToN_IoT repository [37] provides access to the ToN_IoT datasets. The proposed datasets also had label and type features to indicate whether an observation is normal or an attack and the subclasses of attacks for multi-class classification tasks. Various IoT and IIoT sensors across the IIoT network were subjected to nine (9) varieties of cyber-attacks, including Scanning, Denial of service (DoS), DDoS, ransomware, backdoor, data injection, Cross-site Scripting (XSS), password cracking attack, and Man-In-The-Middle (MITM). The dataset's specifications can be found in[37].

However, we are using the data which is mainly generated in Fog layers from Windows 7, Windows 10, and Linux-based OS [Disk activity, Process-scheduling activity, Memory activity]. As stated in [26], the Node-RED utility was utilized to connect sensors to their corresponding backend cloud server in order to generate standard data. Multiple cyber-security incidents were launched against IoT and IIoT sensors of varying types.

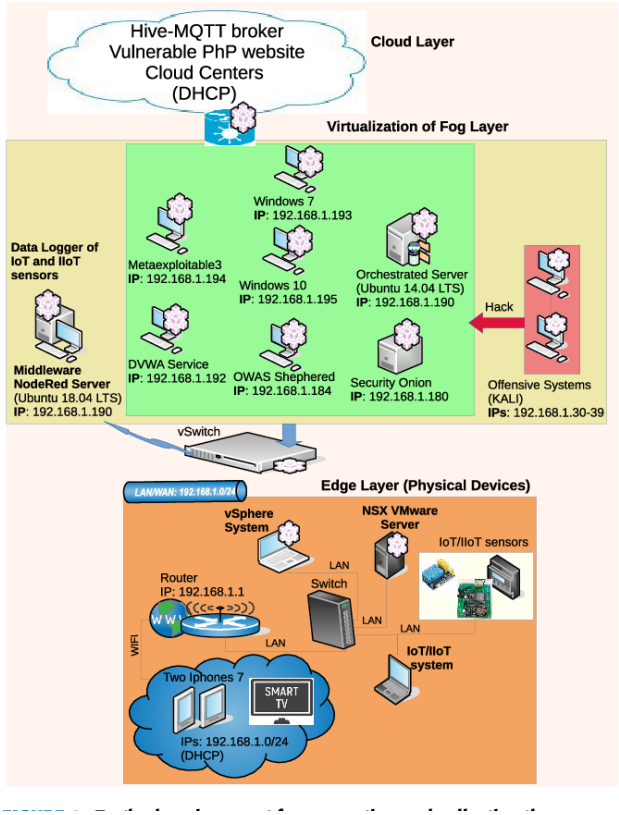

Fig. 1: Testbed environment overview.

The hacking scenarios were designed to exploit either the IP address of Node-RED, public and local Message Queue Telemetry Transport (MQTT) brokers, or the WiFi connections of physical IoT sensors. They used the timestamp field of each well-known attack to designate each vector as normal or the attack. After labeling all the attacks and their categories (attack sub-classes), they added the remaining vectors as either normal or local MQTT brokers. Also, the offensive Kali systems used 10 Kali Linux VMs with static IP addresses and different bash and python scripts to attack vulnerable IIoT network systems and launch attacks against IoT/IIoT services [26].

D. *Methodology*

*1) **Missing value handling**:* To determine the threshold for deleting entries based on the percentage of missing data by column, different studies have used various thresholds. For example, [38] conducted a complete case analysis with pairwise deletion when the percentage of missing data was less than 1.5%. Similarly, [39] applied list-wise deletion and excluded records from analysis if any single value was missing.

On the other hand, [40] proposed ignoring incomplete data when the percentage of missing entries exceeded a certain threshold, while [41] focused their analysis on models without variables that had a substantial amount of missing data.

Furthermore, [42] deleted cases with missing values, resulting in no variables with more than 3% missing values.

Similarly, [43] used listwise deletion due to the relatively small amount of missing data.

In contrast, [44] substituted the 15% missing data with the corresponding column mean, a procedure that preserves information without bias. [45] deleted variables with severe data missing (more than 20% of the total data) and imputed missing data for variables with less than 20% missing values.

Therefore, the threshold for deleting entries based on the percentage of missing data varies across studies, with values ranging from 1.5% to 20%, depending on the specific research context and data analysis approach.

Imputation of missing data is a crucial step in data analysis and ML, as it involves estimating or filling in missing values in a dataset [46]. Ignoring or deleting records with missing values can result in the loss of important information and introduce bias into the analysis [46].

There are various methods for imputing missing data, depending on the characteristics of the dataset and the nature of the missing values.

1) **Mean/median imputation:** One common approach is to use the mean or median value of the variable to fill in the missing data [46]. This method assumes that the missing values are comparable to the observed values.
2) **Hot deck imputation:** Another approach is hot deck imputation, where missing values are filled in by borrowing values from other similar cases in the dataset [46]. This approach aims to preserve the distribution of the original data.
3) **Multiple imputations:** Multiple imputations is a technique that generates multiple plausible imputed datasets and combines the findings from each dataset to obtain accurate estimations [46]. This method takes into account the uncertainty of the imputed values.
4) **Regression imputation:** Regression imputation, on the other hand, involves using regression models to fill in missing data by determining the relationship between the variable with missing values and other variables in the dataset [46]. This approach utilizes the information from related variables to impute the missing values.

The imputation of missing data is a critical step in data analysis and ML. It is important to handle missing data appropriately to avoid loss of information and introducing bias into the analysis [46]. Imputation methods such as mean or median imputation, hot deck imputation, multiple imputations, and regression imputation can help to preserve the integrity of the dataset and obtain accurate estimations [46].

*2) **Data balancing**:* Data balancing is a crucial step in ML, particularly when dealing with imbalanced datasets. Imbalanced datasets occur when the distribution of classes in the dataset is highly skewed, with one class being significantly more prevalent than the other(s). This is a common issue in various domains, including bioinformatics [47], healthcare [48], speech signal analysis [49], tourism [50], cybersecurity [51], and many others sectors.

The importance of data balancing in ML is mainly attributed to two key reasons: to address the problem of data imbalance and to mitigate the bias towards the majority class.

Firstly, data balancing techniques aim to address the problem of data imbalance. Classifiers tend to favor the majority class when ML models are trained on imbalanced datasets, leading to unsatisfactory outcomes for the minority class. This imbalance can lead to several issues, such as reduced accuracy, poor sensitivity, and overfitting to the majority class. By balancing the dataset, the models can achieve better performance by adequately considering the minority class and avoiding skewed predictions.

Secondly, data balancing helps alleviate the bias towards the majority class. ML models' performance may suffer as a result of the bias that the imbalance causes. By balancing the dataset, the models can learn from both classes more effectively, enabling them to generalize better and make more accurate predictions for both the majority and minority classes.

Various approaches and techniques have been proposed to balance imbalanced datasets in ML. These techniques can be broadly categorized into oversampling, undersampling and hybrid methods.

1) **Oversampling:** Oversampling is a technique that increases the number of samples in the minority class. The simplest form of oversampling is random oversampling, which involves repeating randomly selected minority class samples to balance the distribution of classes [52]. A variation of random oversampling is Synthetic Minority Over-sampling Technique (SMOTE), which creates synthetic samples from the minority class by interpolating between existing minority class samples [53].
2) **Undersampling:** Undersampling is a technique that reduces the number of samples in the majority class. The simplest form of undersampling is random undersampling, which involves randomly removing the majority of class samples to balance the distribution of classes [52]. Another variation of random undersampling is Tomek links, which involves removing samples from the majority class that is nearest to the minority class samples [54].
3) **Hybrid Techniques:** Hybrid techniques combine both oversampling and undersampling to balance the distribution of classes. A common hybrid technique is SMOTE-Tomek links, which involves applying SMOTE followed by Tomek links to balance the distribution of classes [55].

Penalty-based techniques are an additional strategy for addressing the imbalance problem in ML. These methods assign higher penalties or costs to misclassifications of the minority class to encourage the classifier to pay more attention to the minority class during training 10.1371/journal.pone.0067863. By adjusting the misclassification costs, the models can prioritize the minority class and make predictions that are more balanced.

So, data balancing is essential in ML to address the problem of data imbalance and mitigate bias towards the majority class. Undersampling, oversampling, hybrid methods, and penalty-based methods are commonly used techniques for data bal-

ancing. The selection of the appropriate technique depends on the specific characteristics of the dataset and the goals of the ML task.

*3) **Sample Correlation coefficient**:* The sample correlation coefficient [56] measures the linear relationship between two variables. Pearson's correlation coefficient, which quantifies linear relationships between continuous variables, is the most popular sample correlation coefficient. Pearson's correlation formula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where x_i and y_i are the i^{th} observation for two variables X and Y, respectively, and $\bar{x}$ and $\bar{y}$ are the means of X and Y, respectively. The value of r varies between -1 and 1, with 1 signifying a perfect positive linear relationship, -1 a perfect negative one, and 0 no linear relationship.

*4) **Predictive Models**:*

- **Support Vector Machine (SVM):**
  The SVM is a supervised ML technique utilized both for classification and regression. It is a well-grounded technique in a theoretical sense as well as being used successfully in many real-world problems [57]. For a classification problem, in order to place new data points into the suitable class,the SVM creates a decision boundary called hyperplane which maps the training observations to higher dimension space through some mapping techniques [58], [59]. It is to be mentioned that in a P dimension space, a hyperplane is an affine subspace of $P-1$ dimension.The SVM takes the extreme vectors or points called support vectors which are used to create the hyperplane. Among infinite possible hyperplanes, the most useful choice is to take the optimal separating hyperplane in order to classify data in the best possible way.The SVM is an algorithm that finds a linear model that maximizes the margin of the hyperplane which provides maximum separation [60]. In fact, the main goal is to create a classifier that can work well on unseen data [60]. In a p-dimensional space, we can define the hyperplane by the following equation according to [61]:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0. \quad (1)$$

  Where β_0, β_1,..., β_p are the parameters and $X = (X_1, X_2, ..., X_p)^T$ is a vector of length p. If equation (3.1) holds, we say that $X = (X_1, X_2, ..., X_p)^T$ is a point on the hyperplane. If X does not satisfy the equation and either

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p > 0.$$

  or

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p < 0.$$

  A separating hyperplane has the characteristic that [61]

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) > 0$$

  for all $i = 1, 2, ..., n$

The observations can be categorized into two distinct classes of $y = y_1, y_2, ..., y_n \in \{-1, 1\}$, where -1 and 1 represents two different classes. Where we are considering n taring observations in p dimensional subspace. It can easily be said that X lies on which side of the hyperplane [57].
If $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} > 0$, then a test observation will be assigned to the class 1 and if $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} < 0$ then any test observation should be categorized in the class -1 [62].

It was mentioned earlier that based on the training observation $x_1, x_2, ..., x_n$, SVM tries to maximize the margin of the hyperplane. The maximal margin hyperplane for the linear classifier is the solution to the following optimization problem [61],

$$\max_{\beta_0, \beta_1, ..., \beta_p, M} M \quad (2)$$

$$Subject\ to \sum_{j=1}^{p} \beta_j^2 = 1, \quad (3)$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geq M,\ \forall i = 1, 2, ..., n \quad (4)$$

Here, M is the margin of the hyperplane which needs to be maximized.

In many cases, there is no separating hyperplane and hence, it is not possible to find a maximal margin classifier. In that case, for the linear classifier, a linear hyperplane is developed to somehow separate the classes. The support vector classifier determines the classification of a test observation based on its position relative to a hyperplane. This hyperplane is carefully selected to effectively divide the majority of training observations into two classes, although it may occasionally misclassify a small number of observations. The selection of the hyperplane is the outcome of the following optimization problem [61]:

$$\max_{\beta_0, \beta_1, ..., \beta_p, M} M \quad (5)$$

$$Subject\ to \sum_{j=1}^{p} \beta_j^2 = 1, \quad (6)$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} \geq M(1 - \varepsilon_i),\ \forall i = 1, 2, ..., n \quad (7)$$

$$\varepsilon \geq 0, \sum_{i=1}^{n} \varepsilon_i \leq C \quad (8)$$

where C is a tuning parameter with non-negative values and the slack variables $\varepsilon_1, \varepsilon_2, ..., \varepsilon_n$ permits each observation to lie on the wrong side of the hyperplane. A test observation $x^* = (x_1^*, x_2^*, ..., x_p^*)$ will be classified by simply determining the sign of $\beta_0^* + \beta_1^* x_1^*, ..., \beta_p^* x_p^*$.
Support vector machines can be used for both separable

and non-separable cases with non-linear decision boundaries [61]. For non-linear decision boundaries, nowadays, kernel tricks are being used in many learning systems. These functions are used mainly to facilitate the computation power of SVM [60]. Kernelizations are well established methods to ease the complexity of parameters [63]. SVM can choose different types of kernels like radial basis function (Gaussian), radial basis function (exponential), linear kernel, Fourier series, polynomial kernel, multi-layer perceptron etc. [60].

We know that the linear support vector classifier can be expressed as,

$$f(x) = \beta_0 + \alpha_i \langle x, x_i \rangle$$

where α_i are the n parameters, $i = 1, 2, .., n$, one representation per training observation.

For the estimation purpose of $\alpha_1, ..., \alpha_n$ and β_0, we need to have the $\binom{n}{2}$ inner products $\langle x_i, x_i' \rangle$ among all pairs of the training observations. When representing the linear classifier $f(x)$ and calculating its coefficients, the only requirement is to utilize the inner product [57]. The inner product can be expressed in some general form as follows,

$$K(x_i, x_i')$$

Where K represents a function referred to as a kernel. A kernel is a specific type of function that measures the similarity between two observations.

If

$$K(x_i, x_i') = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

it is called linear kernel.

It is possible to substitute each occurrence of $\sum_{j=1}^{p} x_{ij} x_{i'j}$ by the quantity

$$K(x_i, x_i') = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$

Then it is referred to as the polynomial kernel where d is the degree of the polynomial.

Another commonly used option is the radial kernel, which is expressed as follows:

$$K(x_i, x_i') = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{ij}')^2)$$

where γ is a positive constant.

For some imaginary data, we can show Support Vectors and hyperplane for separating cases with linear boundaries by the following figure The solid line in the above figure represents a hyperplane. Support vectors are the blue and purple dots on the dashed lines, and the distance of those dots to the hyperplane is shown by arrows. The purple and blue grid defines the decision boundary made by a classifier based on this hyperplane which separates the two classes.

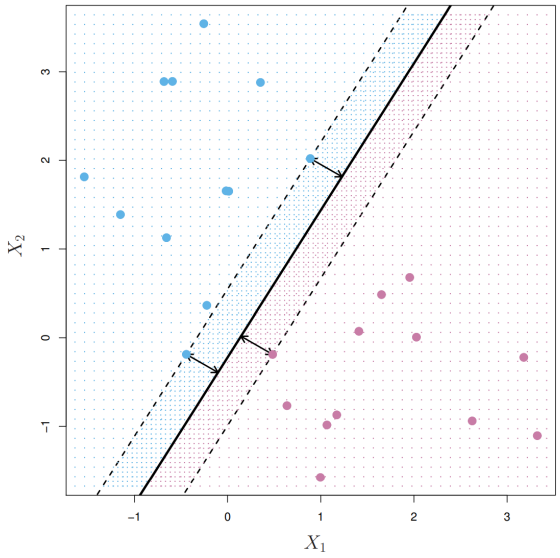

Fig. 2: Support Vectors and Hyperplane.

- **Random Forest (RF) Algorithm:** The RF is an ensemble learning algorithm that creates multiple decision trees using random subsets of training observations [64]. It was introduced by Breiman in 2001 and since then RF became a popular machine-learning method. It reduces overfitting, increases the accuracy and stability of the model as well as tolerates noise and outliers [65]. RF is mainly comprised of a set of tree bases classifiers $\{h(x, \phi_k), k = 1, 2, ...\}$ where ϕ_k are some random vectors which are independent and identically distributed. Each tree within the forest contributes a unit vote towards determining the most prevalent class for a given input x. The basic steps involved in the RF algorithm are as follows:

  1) Select a random subset of data and features
  2) Train a DT on the selected data and features
  3) Repeat steps 1 and 2 multiple times, creating multiple decision trees
  4) Combine the predictions from each DT to produce a final prediction

  After repeating steps 2 and 3 a total of k times, the sequence of classifier $h_1(x), h_2(x), ..., h_k(x)$ is obtained. The ultimate outcome of this system is determined through majority votes [66] and the decision function is,

$$H(x) = argmax_Y \sum_{i=1}^{k} I(h_i(x) = Y)$$

where $H(x)$ represents the combination of multiple decision trees, $h_i(x)$ is the single tree, Y is the response variable, and $I(.)$ serves as the indicator function. In the case of RF, the margin function is employed to quantify the degree to which the average number of at X,Y for the correct class surpasses that for the incorrect class. The margin function can be defined as follows:

$$mg(X,Y) = av_k I(h_k(X) = Y) - max_{j \neq Y} av_k I(h_k(X) = j)$$

As the margin increases, the accuracy of the classification prediction improves, instilling greater confidence in the classification outcome.

In the construction of an RF, the tree is established on a fresh training set through the utilization of random feature selection. The new training set is obtained from the original training set using bagging methods. The reason for using bagging is that it helps to increase accuracy as bagging is related to random features. When an original training set T is given which consists of N samples, the kth training set is obtained by the bagging method from T, which involves drawing N samples with replacement. Each training set T_k contains N samples. As N becomes larger, the probability of a sample not being included in T_k is approximately $(1-1/N)^N$, which converges to e^{-1}. The technique used to estimate the performance of classification using these data is referred to as OOB (Out of Bag) estimation.

The figure below is an example of a RF for classification [65]:

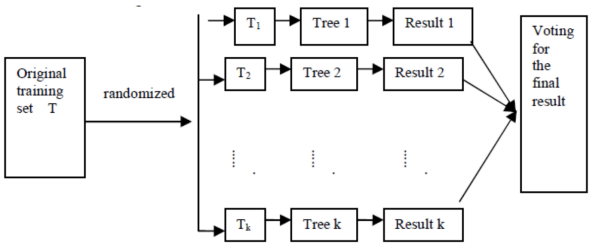

Fig. 3: RF Classifier

- **K-Means Clustering:**K-means clustering is an iterative numerical method that falls under unsupervised ML. It is non-deterministic in nature but known for its simplicity and high speed. Due to these advantages, K-means has proven to be highly effective in practical applications [67]. The K-means clustering algorithm is popular for grouping similar data points into clusters [57]. The algorithm starts with a predefined number of clusters, K, and randomly assigns each data point to one of the K clusters [57]. Let the sets $C_1, C_2, ..., C_K$ refer to clusters and contain the indices of the corresponding observations. These two sets possess two characteristics:
  - Every observation is assigned to a minimum of one of the K clusters. That is, $C_1 \cup C_2 \cup ... \cup C_K = \{1,2,...,n\}$
  - The clusters do not overlap with each other. That is, $C_k \cap C_{k'} = \emptyset$

The concept underlying K-means clustering is that effective clustering is characterized by minimizing the within-cluster variation [57]. The objective function for the K-means clustering algorithm can be expressed as:

$$\min_{C_1,C_2,...,C_K} \{\sum_{k=1}^{K} W(C_K)\}$$

where $W(C_k)$ is the within cluster variation for the kth cluster.

The within-cluster variation can be best described by Euclidean distance which is described below:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

Where $|C_k|$ represents the count of observations within the kth cluster. To put it differently, the within-cluster variation of the kth cluster is calculated by summing the squared Euclidean distances between all the observations within that cluster and then dividing the sum by the total number of observations in the kth cluster. Then the optimization problem for K-means clustering becomes,

$$\min_{C_1,C_2,...,C_K} \{\sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\}$$

There is an algorithm to divide the data into K clusters so that the objective function can be minimized.

Solving this problem is exceedingly challenging due to the vast number of possible ways to partition n observations into K clusters unless K and n are very small. However, there is a ray of hope as a straightforward algorithm can yield a local optimum, which serves as a fairly satisfactory solution to the K-means optimization problem. The algorithm is described below:

1) Assign the initial cluster to the observations by randomly selecting a number between 1 and K for each observation.
2) Continue iterating until the cluster assignments no longer undergo any changes.
   - Calculate the cluster centroid for each of the K clusters. The centroid of the kth cluster is obtained by calculating the mean vector of the p features for the observations within that cluster.
   - Assign each observation to the cluster whose centroid is nearest, with proximity determined using Euclidean distance.

The above algorithm is guaranteed to decrease the objective function value at each step.

- **Self Organizing Map (SOM):** The SOM is a form of artificial neural network utilized for unsupervised learning. It is a non-linear mapping network designed to compute similarities between data in the input layer and represent them in an output layer of interconnected neurons according to spatial constraints. [29]. In contrast to other Artificial Neural Networks, the Kohonen SOM network can be trained through unsupervised learning. The SOM network employs competitive learning to identify data similarities, aggregating them into distinct data classes; it has a feed-forward structure with a single computational layer [29].
The SOM algorithm compares each vector of input data to the codebook vectors of the neurons in order to identify the Best Matching Unit (BMU) [29]. The BMU then modifies its codebook vectors by calculating a weighted average, which brings the vector closer to the input vector. As a training parameter of the model, the learning rate determines the attractiveness between the BMU and the input data vector [29]. To ensure model convergence, this learning rate decreases over time during the training procedure. Additionally, neighboring neurons modify their codebook vectors to better match the input vector, thereby preserving spatial constraints and preserving the topology of the map [29].
Manhattan, Tanimoto, Bray Curtis, Canberra, and Chebyshev distance can be used as the measure of similarity. The most common and effective distance measure is the Euclidean distance [29] as follows:
If X_i with $i = 1,...,n$ the input data vector and codebook vector W_j with $j = 1,...,m$ associated to each neuron. At epoch t,

$$d_j(t) = ||X_i(t) - W_j(t)||, \tag{9}$$

The neuron associated with the codebook vector W_j with the smallest distance to X_i is the winning neuron for that BMU [29]. The distance to the BMU at each epoch t is denoted by:

$$d_c(t) = min_j d_j(t), \tag{10}$$

The whole architecture of SOM can be represented by the following graph as used in [29]:
The SOM algorithm can be mathematically described as follows in updating the spatial neighbors and neurons after founding the BMU [29]:

$$W_j(t+1) = W_j(t) + h_{jc}(t)[X_i(t) - W_j(t)], \tag{11}$$

where, h_{jc} is the neighborhood function. In our study, we used two different types of neighborhood functions: Gaussian and triangular. Gaussian is the most used function in practice [29]:

$$h_{jc} = \alpha(t) exp(\frac{||r_j - r_c||^2}{2\sigma(t)^2}), \tag{12}$$

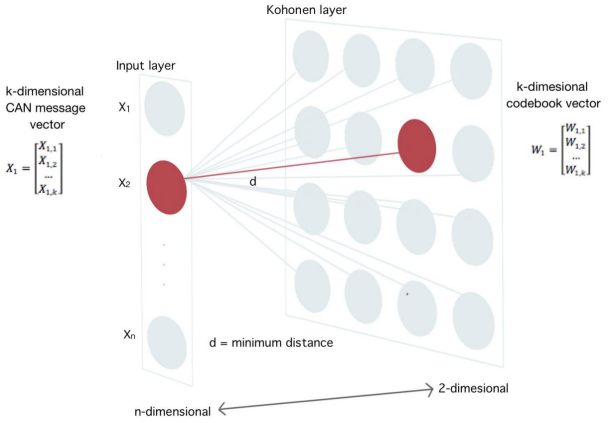

Fig. 4: Kohonen SOM Neural network

where, $\alpha(t)$ is the learning rate, and r_j is the position of the j-th neuron [29].
The SOM algorithm can be broken down into several steps:

1) Initialize the weight vectors of neurons randomly.
2) For each iteration, choose a random input vector $x(t)$ and find the BMU using the Euclidean distance.
3) Update the weight vectors of the BMU and its neighbors using equation 11.
4) Repeat steps 2 and 3 until a stopping criterion is met.

Class	Predicted Positive	Predicted Negative
Class of Attack (Positive)	Predicted Attack as attacks (TP)	Predicted Attacks as Normal (FN)
Class of Normal (Negative)	Predicted Normal as Attacks (FP)	Predicted normal (TN) as Normal

TABLE I: Model evaluation metrics

*5) **Performance Measures Used**:*

- **Accuracy**

Accuracy evaluates a model's predictions. Divide the number of accurately predicted cases by the total number of instances. Accuracy shows how well the model predicts. In skewed datasets, it may not work. Accuracy can be deceiving when the model predicts the majority class. Thus, accuracy and recall are key evaluation metrics, especially in imbalanced datasets, to better understand the model's performance [26].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

- **Precision**

Precision measures a model's positive prediction accuracy. Divide the number of accurately predicted positive instances by the total number of positive instances predicted. High precision means the model has a low false positive rate. It assesses the model's false positive avoidance. In ML and classification tasks, precision

shows the model's positive predictions' dependability and effectiveness [26].

$$Precision = \frac{TP}{TP+FP} \tag{14}$$

- **Recall**

  Recall, sometimes called sensitivity or true positive rate, measures how well a model captures positive cases. The number of accurately predicted positive cases is divided by the dataset's actual positive instances and false negative instances. High recall means the model rarely predicts unpleasant events. It measures the model's positive instance detection accuracy. Recall helps ML and classification models collect all relevant positive instances [26].

$$Recall = \frac{TP}{TP+FN} \tag{15}$$

- **F1-Score**
  The F-score, commonly known as the F1-score, is a performance metric that combines precision and recall. It's the harmonic mean of precision and memory. The F-score balances a model's precision (avoiding false positives) and recall (capturing positive cases). False positives and negatives are taken into consideration, making it useful when the dataset has an imbalance. The F-score goes from 0 to 1, with higher values signifying better performance. Practitioners can use the F-score to assess a model's overall performance by evaluating the precision and recall [26].

$$F1\ Score = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{16}$$

V. Experimental Results

A. *Exploratory Data Analysis and Data Preparation*

This study utilizes five distinct data sets (Windows 7, Windows 10, Linux Disk, Linux Process, and Linux Memory). In every data set, the response variable is a binary value. As our research objective is to apply the best predictive model to each data set, we must first cleanse and process the data. The summary of the original data can be found in the table below.

Data	Number of rows	Number of columns
Windows 7	28,367	135
Windows 10	35,975	127
Linux Disk	160,112	9
Linux Process	160,112	17
Linux Memory	140,112	13

TABLE II: Dataset Information

Before feeding this raw data into the model, this data must be cleansed. We must deal with the data's missing values, redundant variables, data balancing, normalization, and train-test divide.

If a missing value was less than or equal to 1.5%, we discarded the data. We also examined whether there is a column in any of the datasets with zero variability. This column is also eliminated from the dataset. Also for Linux Process data we removed some id columns and created dummy variables for some categorical columns. After creating the dummy variable we deleted the original categorical column from the data (Linux Process Data). After deletion data shape looks like the below:

Data	Number of rows	Number of columns
Windows 7	28,097	79
Windows 10	35,214	106
Linux Disk	160,112	5
Linux Process	158, 707	17
Linux Memory	140,112	9

TABLE III: Transformed Dataset Information

Then, we examined the response variable label counts, or in other words, the number of cyber attack data and non-attack data counts. We can present the subsequent count and percentages using the following pie chart:

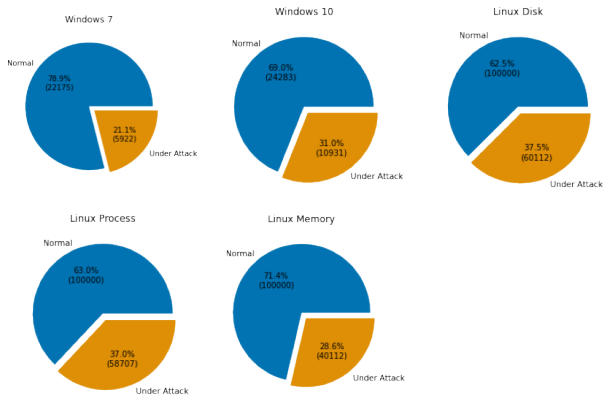

Fig. 5: Pie chart by the attack and normal data.

It is evident from the pie chart that the attack category is much smaller than the normal category. To construct a robust predictive model, it is sufficient to balance the data between these categories; otherwise, the model could be deceptive. Therefore, we utilized the under-sampling technique or another random selection method to select approximately the same number of observations as the under-attack category.

On top of the balance data set, we employed the correlation coefficient procedure among all the predictor variables to identify the perfectly correlated columns and eliminate one of them to get rid of redundant features. Otherwise, it could lead to overfitting, increase the needless complexity of the analysis, and turn the model unreliable overall. Below is displayed the heatmap for all predictors for each data set.

.

For this analysis, we have detected 4 variables pair that perfectly correlated with each other for Windows 7 data.

.

Fig. 6: In search for perfectly correlated variables (Windows 7).

	FEATURE_1	FEATURE_2	CORRELATION
1	Process(_Total) Private Bytes	Process(_Total) Page File Bytes	1.0
3	Network_I(Intel R _Pro_1000MT) Packets Received sec	Network_I(Intel R _Pro_1000MT) Packets Received Unicast sec	1.0
5	Network_I(Intel R _Pro_1000MT) Packets Received Unicast sec	Network_I(Intel R _Pro_1000MT) Packets Received sec	1.0
7	Memory Available KBytes	Memory Available Bytes	1.0

Fig. 7: Perfectly correlated variables (Windows 7).

So, Process(_Total) Private Bytes, Network_I(Intel R _Pro_1000MT) Packets Received sec, Network_I(Intel R _Pro_1000MT) Packets Received Unicast sec, Memory Available KBytes variables deleted from the data.

Using a similar approach we can detect the perfectly correlated for Windows 10 data as well.

.

.

For Windows 10 data these variables were removed because of perfect correlation: Network_I(Intel R _82574L_GNC) Packets Received Unicast sec, Processor_pct_ C1_Time, Network_I(Intel R _82574L_GNC) Packets Received sec, Memory Available KBytes, Process_Private_Bytes.

For the Linux Disk and memory data, none of the independent variables are precisely correlated, but we have nonetheless balanced the data. Although, we removed 3 columns for Linux Process data because of perfect correlation.

.

.

.

The table IV represents the total number of rows and columns for each data set after cleaning and applying different statistical methods. We normalized each column and sliced the clean data into train and test sets using an 80%-20% split in order to implement the selected machine-learning models.

Fig. 8: In search for perfectly correlated variables (Windows 10).

	FEATURE_1	FEATURE_2	CORRELATION
1	Processor_pct_ C1_Time	Processor_pct_ Idle_Time	1.0
3	Process_Private_Bytes	Process_Page_File Bytes	1.0
5	Network_I(Intel R _82574L_GNC) Packets Received sec	Network_I(Intel R _82574L_GNC) Packets Received Unicast sec	1.0
7	Network_I(Intel R _82574L_GNC) Packets Received Unicast sec	Network_I(Intel R _82574L_GNC) Packets Received sec	1.0
9	Memory Available KBytes	Memory Available Bytes	1.0

Fig. 9: Perfectly correlated variables (Windows 10).

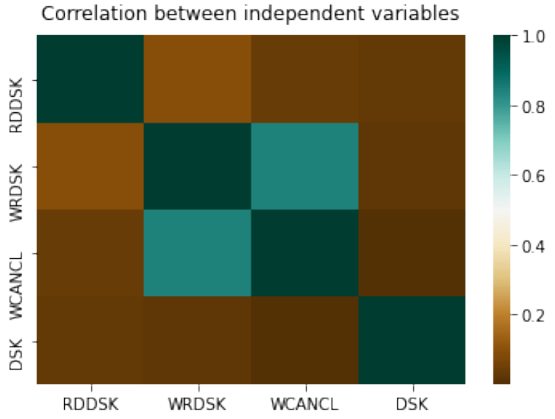

Fig. 10: In search for perfectly correlated variables (Linux Disk).

Data	Number of rows	Number of columns
Windows 7	11,909	75
Windows 10	21,858	101
Linux Disk	120,222	5
Linux Process	117,417	14
Linux Memory	80,222	9

TABLE IV: Cleaned data Information

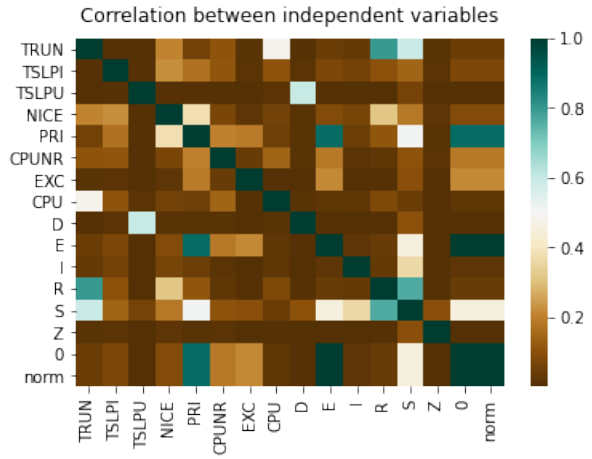

Fig. 11: In search for perfectly correlated variables (Linux Process).

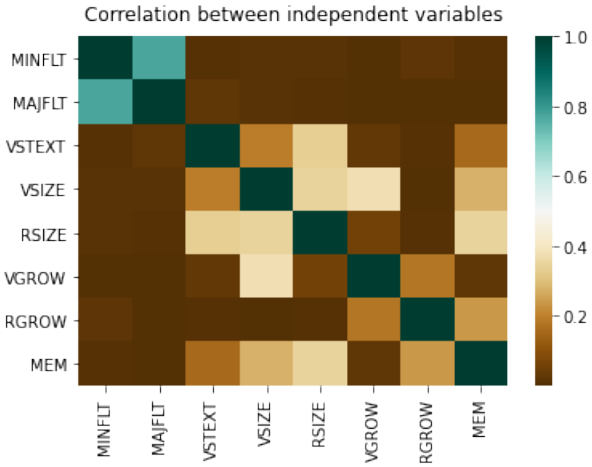

Fig. 12: In search for perfectly correlated variables (Linux Memory).

B. Model Performance Evaluation

GridSearchCV, a method for methodically searching a parameter grid to identify the best hyperparameters for a model, was used to discover the best model for our classification problem.

We tested alternative regularization parameter C values and kernel types (rbf, poly, sigmoid, linear) for the SVM model. The GridSearchCV algorithm evaluated each parameter combination with 10 folds of cross-validation. The models were scored on accuracy.

The RF model's hyperparameters included the number of trees (n_estimators), the maximum number of features considered for each split (max_features), the maximum tree depth (max_depth), and the split quality criterion (gini, entropy). Again, accuracy was used to score each hyperparameter combination in a 10-fold cross-validation.

In addition to the SVM and RF models, we used GridSearchCV for the SOM and K-Means clustering.

We searched a parameter grid with the sigma value, learning rate, and neighborhood function for the SOM model. The learning rate determines the weight update step size, the sigma value governs the neighborhood function width, and the neighborhood function defines the influence of nearby neurons during training. GridSearchCV used 10-fold cross-validation to assess each parameter combination's accuracy.

For the KMeans model, we examined the number of initializations (n_init), the initialization method (k-means++, random), and the clustering strategy (auto, full, elkan). The GridSearchCV method again compared hyperparameter combinations using 10-fold cross-validation and the accuracy metric.

We found the optimal hyperparameters and model after grid-searching each model. We then trained the best model on training data and tested it on test data. Grid search process execution times were also recorded. This systematic grid search allowed us to effectively investigate alternative hyperparameter combinations and find the optimal classification model. To improve model accuracy and performance, we tuned hyperparameters.

*1) **Windows 10**:* We measured split quality using an entropy-based criterion and set the tree depth to 25 for the RF model. We utilized 100 decision trees (n_estimators) to build the forest and limited each split to the square root of the total number of features. This hyperparameter combination detected intrusions best.

We used a linear kernel and set C to 50 for the SVM model. These SVM settings accurately identified intrusions.

The SOM model used a triangle neighborhood function with a sigma value of 2 and a learning rate of 0.1. These parameters enabled the SOM to map incoming data into a topological grid and detect intrusions based on their spatial relationships.

Finally, we employed the 'auto' algorithm for K-means clustering. We used k-means++ to initialize centroids and ran 50 iterations (n_init) to determine the best clustering solution. This combination produced the highest accurate intrusion clustering findings.

The performance of the selected models on test data is presented in the following table:

Model	Accuracy	Precision	Recall	F-1 Score
RF	0.986	0.974	0.998	0.986
SVM	0.968	0.949	0.990	0.969
SOM	0.710	0.655	0.889	0.754
K-means	0.839	0.878	0.839	0.834

TABLE V: Model Performance Comparison (Windows 10)

As shown in Table 1, the results of our intrusion detection analysis emphasize the performance of various models in terms of accuracy, precision, recall, and F-1 score. RF and SVM had the greatest accuracy scores among the evaluated models, at 0.986 and 0.968, respectively. The high precision values of 0.974 and 0.949 attained by these models indicate their capacity to correctly classify intrusions. Similarly, RF and SVM achieved high recall scores of 0.998 and 0.990, respectively, indicating their ability to detect intrusions accurately. As a

result, these models achieved remarkable F-1 scores of 0.986 and 0.969, indicating a performance that is balanced between precision and recall.

It is essential to note, however, that the SOM and K-means models performed less well than RF and SVM. The SOM model achieved an accuracy rating of 0.710, with precision, recall, and F-1 ratings of 0.655, 0.889, and 0.754, respectively. Compared to the other models, the SOM model may have difficulty accurately classifying intrusions, as indicated by these results. Likewise, the K-means model obtained an accuracy score of 0.839, as well as precision, recall, and F-1 scores of 0.878, 0.839, and 0.834, respectively. While the K-means model performed better than the SOM model, its accuracy and F-1 scores were lower than those of RF and SVM.

The results of this study indicate that RF and SVM are the most effective intrusion detection models. In terms of accuracy, precision, recall, and F-1 score, these models consistently outperformed the SOM and K-means models. RF and SVM are suitable candidates for real-world IDS due to their high accuracy and balanced performance.

*2) **Windows 7**:* For Windows 7 data, the selected models after the grid search are presented below:

- RF: criterion = entropy, max_depth = 15, max_features = log2, n_estimators = 50
- SVM: C= 50, kernel= linear
- SOM: learning_rate = 0.05, neighborhood_function = gaussian, sigma = 1
- K-means: algorithm = auto, init = k-means++, n_init = 50

Model	Accuracy	Precision	Recall	F-1 Score
RF	0.994	0.994	0.993	0.994
SVM	0.954	0.968	0.939	0.953
SOM	0.623	0.589	0.797	0.677
K-means	0.540	0.551	0.539	0.510

TABLE VI: Model Performance Comparison (Windows 7)

The success of the four models, RF, SVM, SOM, and K-means, was evaluated using various metrics, including precision, recall, F-1 score, and accuracy. With an accuracy, precision, recall, and F-1 score of 0.99, the RF model obtained the highest result. This demonstrates that the RF model accurately classified intrusion instances with a high degree of precision and recall. With an accuracy of 0.954, a precision of 0.968, a recall of 0.939, and an F-1 score of 0.953, the SVM model also performed admirably. The performance of the SOM and K-means models was inferior, with the SOM model achieving an accuracy of 0.623, precision of 0.589, recall of 0.797, and F-1 score of 0.677, and the K-means model achieving an accuracy of 0.540, precision of 0.551, recall of 0.539, and F-1 score of 0.510. The RF and SVM models outperformed the SOM and K-means models in accurately detecting and classifying intrusions in the dataset, according to these results.

*3) **Linux Disk activity**:* Selected models after hyperparameter tuning are:

- RF: criterion = gini, max_depth = 5, max_features = log2, n_estimators = 50
- SVM: C= 50, kernel= linear
- SOM: learning_rate = 0.5, neighborhood_function = gaussian, sigma = 2
- K-means: algorithm = auto, init = k-means++, n_init = 50

Model	Accuracy	Precision	Recall	F-1 Score
RF	0.537	0.920	0.081	0.148
SVM	0.522	0.984	0.045	0.0873
SOM	0.520	0.553	0.210	0.305
K-means	0.504	0.751	0.504	0.343

TABLE VII: Model Performance Comparison (Linux Disk)

The RF model obtained the highest accuracy of 0.537, indicating that 53.7% of instances were correctly classified. However, its precision, recall, and F-1 score were comparatively low, indicating that it had difficulty detecting and classifying intrusions accurately. The SVM model's accuracy was slightly lower at 0.522, but its precision was high at 0.984, indicating a low rate of false positives. However, its recall and F-1 scores were very low, indicating that it had difficulty detecting true positives. The SOM and K-means models had comparable accuracy rates of 0.520 and 0.504, respectively, but their precision, recall, and F-1 scores were relatively low, indicating inadequate intrusion detection performance. Overall, the models demonstrated varying degrees of accuracy in detecting and classifying intrusions, with the RF model demonstrating the highest accuracy but the lowest precision, and the SVM model demonstrating the highest precision but the lowest recall.

*4) **Linux Process-scheduling activity**:* Selected models are:

- RF: criterion = gini, max_depth = 15, max_features = auto, n_estimators = 50
- SVM: C= 50, kernel= rbf
- SOM: learning_rate = 0.5, neighborhood_function = gaussian, sigma = 1
- K-means: algorithm = auto, init = k-means++, n_init = 50

Model	Accuracy	Precision	Recall	F-1 Score
RF	0.737	0.757	0.700	0.727
SVM	0.615	0.722	0.373	0.492
SOM	0.578	0.552	0.833	0.664
K-means	0.422	0.394	0.422	0.382

TABLE VIII: Model Performance Comparison (Linux Process)

With an accuracy of 0.737, the RF model correctly classified 73.7% of instances. It also had the highest precision and F-1 score, indicating that it accurately identified and classified intrusions with a low false positive rate. The SVM model had a lower accuracy of 0.615 but a comparatively high precision of 0.722, suggesting a lower false-positive rate. However, its recall and F-1 scores were comparatively low, indicating that

it had difficulty detecting true positives. With an accuracy of 0.578 and a recall of 0.833, the SOM model demonstrated a strong ability to detect intrusions. However, its precision and F-1 score were inferior to those of competing models. The K-means model had the lowest overall performance, with an accuracy of 0.422, as well as low precision, recall, and F-1 score. Overall, across all evaluation metrics, the RF model demonstrated the highest performance.

*5) **Linux Memory activity**:* Selected models are:

- RF: criterion = entropy, max_depth = 25, max_features = log2, n_estimators = 200
- SVM: C= 50, kernel= rbf
- SOM: learning_rate = 0.1, neighborhood_function = triangle, sigma = 6
- K-means: algorithm = full, init = random, n_init = 100

Model	Accuracy	Precision	Recall	F-1 Score
RF	0.979	0.965	0.994	0.979
SVM	0.611	0.602	0.653	0.637
SOM	0.598	0.613	0.532	0.570
K-means	0.534	0.595	0.534	0.446

TABLE IX: Model Performance Comparison (Linux Memory)

Among the four models, the RF model had the highest performance, with an accuracy of 0.979, precision of 0.965, recall of 0.994, and F-1 score of 0.979. This indicates that the model classified the majority of instances accurately and had a low rate of false positives and false negatives. The SVM model's accuracy was 0.611, while its precision, recall, and F-1 score were moderate. The performance of the SOM and K-means models was even worse, with accuracy, precision, recall, and F-1 scores ranging from 0.534 to 0.598. Overall, the RF model outperformed the competition, making it the most dependable option for intrusion detection.

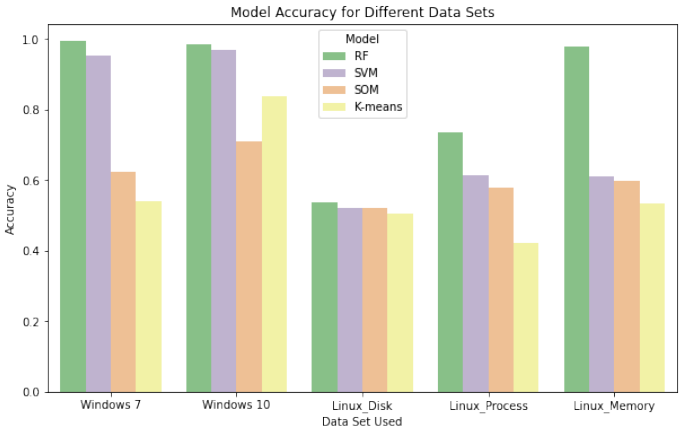

Fig. 13: Model accuracy plot.

The above accuracy plot compares the detection accuracy of various intrusion detection methods across platforms and data types. Upon analyzing the results, it is clear that the RF model consistently outperformed the other models in the majority of instances. After looking over the data, it's clear that the RF model is superior to the others in the vast majority of cases.

An accuracy of 0.994 was reached by the RF model when applied to Windows 7 data, demonstrating its superiority in correctly labeling cases. Again confirming its robustness in detecting intrusions, the RF model attained an accuracy of 0.986 when applied to data from Windows 10. The SVM model, in comparison, demonstrated reduced accuracy (0.954 and 0.968, respectively, for Windows 7 and Windows 10 data). Although the SVM model's strength is in its capacity to deal with complicated data with non-linear bounds, it nevertheless did a respectable job.

All models performed poorly when evaluated on Linux data compared to Windows data. When comparing accuracy on the Linux Disk, Linux Process, and Linux Memory datasets, the RF model was still the most successful.

The SOM and K-means models provide illustrative examples of the benefits of the unsupervised ML approach. While their accuracy was lower than that of supervised models, unsupervised models might perform intrusion detection without the use of labeled data and so might be the most effective tool for exposing new forms of attack. The SOM model had the highest accuracy among unsupervised models for the Windows 7 and Linux Disk datasets, while the K-means model had the highest accuracy for the Windows 10 data.

VI. Conclusions

Comparing the efficacy of supervised and unsupervised models, supervised models typically perform better [68], [69], [70], [71]. Therefore, a direct comparison between supervised and unsupervised algorithms should not be possible in most cases. However, out of pure curiosity, we applied both types of algorithms to our data and discovered promising results for the models.

Our research aimed to identify the best ML model for intrusion detection in ToN_IoT data, specifically focusing on Windows and Linux OS. Throughout our analysis, we evaluated several models, including RF, SVM, SOM, and K-means clustering. Based on our findings, RF emerged as the overall best model for detecting intrusion in sensor data, as supported by previous research [26]. This finding was consistent across various OS and data types, highlighting the robustness and effectiveness of RF in intrusion detection tasks.

However, we also observed that unsupervised models like SOM and K-means offer a viable alternative for intrusion detection in scenarios where labeled data is limited or unavailable, or when facing novel attacks. These unsupervised models leverage patterns and similarities within the data to identify potential intrusions, making them particularly useful in situations where prior knowledge about attack patterns may be lacking.

Looking ahead, future research in this domain could explore other supervised and unsupervised methods, as well as experiment with different hyperparameter settings to optimize the performance of intrusion detection models. Additionally,

further investigations could be conducted to evaluate the performance of these models in real-world deployment scenarios and consider the scalability and computational requirements of each approach.

In summary, our study contributes to the domain of intrusion detection in ToN_IoT data by highlighting the strengths of RF as the preferred model for detecting intrusions across different OS. Furthermore, we recognize the importance of unsupervised models like SOM and K-means in situations where labeled data is scarce, offering a promising avenue for upcoming research. Ultimately, the findings of this study can guide the development of effective IDS for securing IoT networks and protecting against potential threats.

REFERENCES

[1] Jensen K. Philipsen K. Haug A. Stentoft, J. Drivers and barriers for industry 4.0 readiness and practice: A sme perspective with empirical evidence. 2019.

[2] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[3] Xian Guo, Keyu Chen, An Yang, and Zhanhui Gang. Research on industrial iot security based on deep learning. *Journal of Internet Technology*, 24(3):727–744, 2023.

[4] R. Chalapathy and S. Chawla. Deep learning for anomaly detection: a survey. 2019.

[5] Nelson B. Sears R. Joseph A. Tygar J. Barreno, M. Can machine learning be secure? 2006.

[6] Waoo A. Jain, J. An artificial neural network technique for prediction of cyber-attack using intrusion detection system. *JAIMLNN*, pages 33–42, 2023.

[7] Andrade R. Praça I. Sousa O. Maia E. Vitorino, J. A comparative analysis of machine learning techniques for iot intrusion detection. pages 191–207, 2022.

[8] C. Wang, B. Wang, H. Liu, and H. Qu. Anomaly detection for industrial control system based on autoencoder neural network. *Wireless Communications and Mobile Computing*, 2020:1–10, 2020.

[9] Y. Shin and K. Kim. Comparison of anomaly detection accuracy of host-based intrusion detection systems based on different machine learning algorithms. *International Journal of Advanced Computer Science and Applications*, 11, 2020.

[10] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009.

[11] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.

[12] Ranjit Panigrahi and Samarjeet Borah. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, 7(3.24):479–482, 2018.

[13] Nour Moustafa. A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets. *Sustainable Cities and Society*, 72:102994, 2021.

[14] Tim M. Booij, Irina Chiscop, Erik Meeuwissen, Nour Moustafa, and Frank T. H. den Hartog. Ton_iot: The role of heterogeneity and the need for standardization of features and attack types in iot network intrusion data sets. *IEEE Internet of Things Journal*, 9(1):485–496, 2022.

[15] Nour Moustafa, Marwa Keshky, Essam Debiez, and Helge Janicke. Federated ton_iot windows datasets for evaluating ai-based security applications. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 848–855, 2020.

[16] Abdullah Alsaedi, Nour Moustafa, Zahir Tari, Abdun Mahmood, and Adnan Anwar. Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020.

[17] Javed Ashraf, Marwa Keshk, Nour Moustafa, Mohamed Abdel-Basset, Hasnat Khurshid, Asim D. Bakhshi, and Reham R. Mostafa. Iotbotids: A novel statistical learning-enabled botnet detection framework for protecting networks of smart cities. *Sustainable Cities and Society*, 72:103041, 2021.

[18] Nour Moustafa, Mohiuddin Ahmed, and Sherif Ahmed. Data analytics-enabled intrusion detection: Evaluations of ton_iot linux datasets. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 727–735, 2020.

[19] Nour Moustafa. A systemic iot-fog-cloud architecture for big-data analytics and cyber security systems: A review of fog computing, 2019.

[20] M. Govindarajan. Hybrid intrusion detection using ensemble of classification methods. *International Journal of Computer Network and Information Security*, 6:45–53, 2014.

[21] Adebowale Ajayi and Idowu S.A. Comparative study of selected data mining algorithms used for intrusion detection. 07 2013.

[22] S. Naseer and Y. Saleem. Enhanced network intrusion detection using deep convolutional neural networks. *Ksii Transactions on Internet and Information Systems*, 12, 2018.

[23] M. Alsoufi, S. Razak, M. Siraj, I. Nafea, F. Ghaleb, F. Saeed, and M. Nasser. Anomaly-based intrusion detection systems in iot using deep learning: a systematic literature review. *Applied Sciences*, 11:8383, 2021.

[24] M. Goldstein and S. Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *Plos One*, 11:e0152173, 2016.

[25] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. 2003.

[26] Abdullah Alsaedi, Nour Moustafa, Zahir Tari, Abdun Mahmood, and Adnan Anwar. Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020.

[27] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Feature analysis for ml-based iiot intrusion detection. *CoRR*, abs/2108.12732, 2021.

[28] Ziadoon Kamil, Y. Robiah, Nazrulazhar Bahaman, Salama Mostafa, and Cik Feresa Mohd Foozy. Benchmarking of machine learning for anomalybased intrusion detection systems in the cicids2017 dataset. *IEEE Access*, 02 2021.

[29] Vita Santa Barletta, Danilo Caivano, Antonella Nannavecchia, and Michele Scalera. Intrusion detection for in-vehicle communication networks: An unsupervised kohonen som approach. *Future Internet*, 12(7), 2020.

[30] A. Alsaedi, N. Moustafa, Z. Tari, A. N. Mahmood, and A. Anwar. $\mathrm{Ton}_i ot telemetry dataset : a new generation dataset of iot and iiot for data-driven intrusion detection systems. IEEE Access, 8 : 165130 --165150, 2020.$

[31] H. Karamollaoglu, İ. Yücedağ, and İ. A. Doğru. A hybrid pca-mao based lstm model for intrusion detection in iot environments. 2022.

[32] T. Le, H. Kim, H. Kang, and H. Kim. Classification and explanation for intrusion detection system based on ensemble trees and shap method. *Sensors*, 22:1154, 2022.

[33] N. Moustafa, M. Ahmed, and S. Ahmed. Data analytics-enabled intrusion detection: evaluations of $\mathrm{ton}_i ot linux datasets. 2020 IEEE 19th International Conference on Trust, Security and Privacy in C$

[34] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. d. Hartog. $\mathrm{Ton}_i ot : the role of heterogeneity and the need for standardization of features and attack types in iot networ$ $485 - -496, 2022.$

[35] Moustafa N. Ding W. Ding W. Abdel-Basset, M. Supervised deep learning for secure internet of things. pages 131–166, 2021.

[36] Essop I. Mantas G. Porfyrakis K. Ribeiro J. Rodriguez J. Zachos, G. An anomaly-based intrusion detection system for internet of medical things networks. *Electronics*, 10:2562, 2021.

[37] N. Mustafa. Ton-iot dataset [online]. https://cloudstor.aarnet.edu.au/plus/s/ds5zW91vdgjEj9i, 2020.

[38] Leão T. Soares P. Severo M. Moniz M. Lucas R. … Barros H. Leite, A. A case-control study of contextual factors for sars-cov-2 transmission. *Front. Public Health*, 9, 2021.

[39] MacDonald R. Bronskill S. Schull M. Singh, J. Incidence and predictors of critical events during urgent air-medical transport. *Canadian Medical Association Journal*, 181:579–584, 2009.

[40] Ma X. Qin, H. Data analysis approaches of interval-valued fuzzy soft sets under incomplete information. *IEEE Access*, 7:3561–3571, 2019.

[41] Vedlitz A. Zahran S. Alston L. Lubell, M. Collective action, environmental activism, and air quality policy. *Political Research Quarterly*, 59:149–160, 2006.

[42] Prati G. Cicognani E. Prati, G. Coping strategies and collective efficacy as mediators between stress appraisal and quality of life among rescue workers. *Sport, Exercise, and Performance Psychology*, 1:84–93, 2011.

[43] Bahr S. Hoffmann J. Harmon E. Dorius, C. Parenting practices as moderators of the relationship between peers and adolescent marijuana use. *Journal of Marriage and Family*, 66:163–178, 2004.

[44] AZPRA E. Zarraluqui V. GAY C. Bravo, J. Some variations of the rainfall in mexico city from 1954 to 1988 and their statistical significance. *ATM*, 27:367–376, 2015.

[45] Mei Z. Wang Y. Shou X. Zeng R. Chen Y. ... Liu Q. Chen, J. A nomogram to predict in-hospital mortality in patients with post-cardiac arrest: a retrospective cohort study. *Polish Archives of Internal Medicine*, 2022.

[46] Seabloom E. Jones M. Schildhauer M. Borer, E. Some simple guidelines for effective data management. *Bulletin of the Ecological Society of America*, 90:205–214, 2009.

[47] Dunbrack R. L. Wei, Q. The role of balanced training and testing data sets for binary classifiers in bioinformatics. *PLoS ONE*, 8:e67863, 2013.

[48] Fang Y. Wu, Y. Stroke prediction with machine learning methods among older chinese. *IJERPH*, 17:1828, 2020.

[49] Yaseen M. A. Aleesa A. Thanoun, M. Y. Development of intelligent parkinson disease detection system based on machine learning techniques using speech signal. *International Journal on Advanced Science, Engineering and Information Technology*, 11:388, 2021.

[50] Hornillos N. B. Val P. B. Maraña P. Cosío Á. H. Castillo, S. B. Machine learning to predict recommendation by tourists in a spanish province. *Int. J. Info. Tech. Dec. Mak.*, 21:1297–1320, 2022.

[51] Ali R. H. Abideen Z. U. Khan T. A. Kouatly R. Hassan, I. U. Significance of machine learning for detection of malicious websites on an unbalanced dataset. *Digital*, 2:501–519, 2022.

[52] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[53] Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. San Francisco: Morgan Kaufmann Publishers, 2006.

[54] David R Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE transactions on Systems, Man, and Cybernetics*, 2(3):408–421, 1972.

[55] Gustavo Enrique da Silva Alves de Batista, Ronaldo Cristiano Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, 6(1):20–29, 2004.

[56] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London*, 186:343–414, 1895.

[57] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[58] Tom Howley and Michael G Madden. The genetic kernel support vector machine: Description and evaluation. *Artificial intelligence review*, 24:379–395, 2005.

[59] Vladimir N Vapnik. Statistical learning theory hardcover, 1998.

[60] Vahid Hooshmand Moghaddam and Javad Hamidzadeh. New hermite orthogonal polynomial kernel and combined kernels in support vector machine classifier. *Pattern Recognition*, 60:921–935, 2016.

[61] Gareth, Daniela James, Trevor Witten, Robert Hastie, and Tibshirani. An introduction to statistical learning with applications in r. *ISLR*, 2021.

[62] Li Li et al. *Selected applications of convex optimization*, volume 103. Springer, 2015.

[63] Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[64] Mariana Belgiu and Lucian Drăguţ. Random forest in remote sensing: A review of applications and future directions. *ISPRS journal of photogrammetry and remote sensing*, 114:24–31, 2016.

[65] Baoxiang Liu, Maode Ma, and Jincai Chang. *Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16, 2012, Revised Selected Papers*, volume 7473. Springer, 2012.

[66] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[67] Shi Na, Liu Xumin, and Guan Yong. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *2010 Third International Symposium on intelligent information technology and security informatics*, pages 63–67. Ieee, 2010.

[68] Tsai Y. Wang S. Yang M. Cheng, J. Segflow: Joint learning for video object segmentation and optical flow. 2017.

[69] Madhamshettiwar P. Davis M. Ragan M. Maetschke, S. Supervised, semi-supervised and unsupervised inference of gene regulatory networks. *Briefings in Bioinformatics*, 15:195–211, 2013.

[70] Xiang T. Wang Y. Pontil M. Gong S. Huang T. ... Tian Y. Peng, P. Unsupervised cross-dataset transfer learning for person re-identification. 2016.

[71] Ma C. Liu W. Li H. Song, Y. Unsupervised deep tracking. 2019.