Capítulo 3

Intrusion detection with Machine Learning in Internet of Things and Fog Computing: problems, solutions and research

Cristiano Antonio de Souza, Carlos Becker Westphall and Renato Bobsin Machado

Abstract

Intrusion detection is one of the key points in computer security, and it aims to identify attempted attacks by unauthorized users. Several researches are being developed to solve security problems in environments involving the Internet of Things, Fog Computing, and Cloud Computing. This mini-course has a theoretical and practical profile, aims to describe aspects of the context of intrusion detection in IoT and Fog Computing, presents Machine Learning techniques commonly used in intrusion detection, expose state-of-the-art approaches, and present some results obtained in developed research.

1.1. Introduction

With the development of technological resources and the popularization of the Internet, there has been significant growth in the number of computational applications. Faced with this new technological context, difficulties have arisen in maintaining the security of applications and data, given that the techniques for exploiting vulnerabilities in these computational infrastructures are constantly being improved to acquire access to systems and obtain and use improperly sensitive information.

Malicious users can exploit vulnerabilities in computer systems to carry out illicit activities. The attackers' main motivation is to obtain privileged digital content that can bring some benefit to the attacker and/or cause significant damage to the target of the attacks.

Currently, the Internet of Things (IoT) is spreading in all areas that apply computational resources. IoT devices allow everyday objects to be connected to the Internet, computers, and smartphones [Atzori et al. 2010]. The idea is to increasingly unite the physical and digital worlds by communicating objects with other devices, data centers, and clouds.

IoT devices have limited resources [Atzori et al. 2010]. There is a need to transfer, via the Internet, the data generated by these devices to process and store them in a computational center with greater capacity [Miorandi et al. 2012]. Many IoT applications use Cloud Computing to process and store data [Al-Fuqaha et al. 2015]. However, with the growth of IoT, applications began to deal with generating large amounts of data. Consequently, requiring heavy computational resources like bandwidth [Bonomi et al. 2012]. This large amount of data results in network congestion in the communication of IoT devices with data centers of cloud computing [Roman et al. 2018]. Fog computing provides services closer to the end devices [Bonomi et al. 2012, Roman et al. 2018]. Performing temporary data processing and storage close to IoT devices decreases the traffic sent to the cloud [Roman et al. 2018]. In addition, it allows applications that need real-time processing to obtain a faster response [Al-Fuqaha et al. 2015]. In this way, they do not send data from devices to the cloud.

IoT allows various physical objects to see, hear, feel, think, and communicate in the environment in which they are inserted. In this way, each object can share information with other objects and make decisions to perform certain tasks [Atzori et al. 2010]. As a consequence of this scenario, large amounts of information of the most varied natures are processed. Much of this data is confidential and private. Sensitive user data is collected, processed, transmitted, and stored via fog computing and IoT components [Roman et al. 2018]. In this context, attackers may be able to compromise a smart home system, for example, and discover confidential information about the family's habits, such as the time that residents sleep or that the house is empty, among others. This information can later be used to cause serious harm to victims [Ni et al. 2018].

The resource constraints of IoT devices make them susceptible to flaws and malicious data integrity attacks [Neshenko et al. 2019]. This can lead to unreliability and sometimes system collapse. One of the main objectives of an attack on an IoT network is to disrupt the availability of data sent from IoT devices to applications. This interruption can be achieved in several ways, such as overloading devices with information requests or compromising the network structure by dropping packets [Roman et al. 2018].

Smart environments are becoming real and possible through IoT. However, as mentioned above, they are also not free from security threats and vulnerabilities. In this context, in parallel with technological growth, there are also difficulties in maintaining the security of applications and computational infrastructures, considering that vulnerabilities also increase with the growth in the number of available services. A major incident involving IoT devices occurred in October 2016. An attack involving IoT devices via botnet Mirai against service provider Dyn took offline for several hours, hundreds from sites, including Twitter, Netflix, Reddit, and GitHub [Tanaka and Yamaguchi 2017].

This makes special security techniques indispensable in modern computer systems. According to [Roman et al. 2018], security is one of the biggest challenges to ensure an ideal IoT and fog computing environment, where devices can take advantage of the services provided by the paradigm. Intrusion detection is one of the key security points to identify attempted attacks.

There are several state-of-the-art approaches to detecting intrusions in an IoT environment. Some works focus on signature detection. These approaches fail to detect new

attacks or variations of known attacks [Arshad et al. 2019]. In addition, specification-based methods have also been proposed. However, these approaches require a human expert to specify the expected behavior of the network. Finally, other approaches have proposed anomaly-based methods to detect intrusions [Labiod et al. 2022, Rey et al. 2022, Souza et al. 2022a]. Anomaly detection considers that all abnormal behavior is an intrusion and thus can detect new attacks or variations of known attacks. Machine Learning (ML) methods are commonly applied in this context [Boukerche et al. 2007]. However, anomaly-based approaches often require features that IoT nodes lack. [Ahmad et al. 2021] highlight the resources consumed by complex models and the need for lightweight IDS for IoT. Thus, most of the studies work with fog computing.

Several state-of-the-art approaches focus on anomaly methods for binary detection (attack or non-attack) [Albdour et al. 2020, Kumar et al. 2021a, Rey et al. 2022]. They can detect an intrusion, but not the type or category of attack. In this context of intrusion detection, the approach must identify the attack category so that more specific countermeasures can be implemented for the given type of threat. Identifying the type or category of the attack is also important for the decision-making process of the person responsible for the network [Souza et al. 2022b]. Several multiclass approaches have been proposed in state of the art. However, multiclass detection approaches are generally more complex, have a higher computational cost, and have lower accuracy rates than binary methods [Nguyen et al. 2019]. This is justified by the difficulties in identifying specific types of attacks [Diro and Chilamkurti 2018, Kumar et al. 2020a].

The prohibitive cost is another important point, as the resource constraints present in IoT and Fog computing environments limit the design of robust approaches. Robust and slow multi-class analysis performed on IoT/Fog can overload the device and cause network flow delay [Nguyen et al. 2019]. Furthermore, the use of attribute selection and class balancing techniques, useful to improve detection performance, tend to increase the training cost of the approaches. Therefore, intrusion detection in these environments has challenges and research opportunities.

This mini-course presents the main concepts in this context, and several machine-learning techniques used to detect intrusions are addressed. From this, practical activity is proposed for the execution of experiments with the studied techniques. Subsequently, it is presented in the state of the art how these techniques have been applied to detect intrusions in the IoT environment. Finally, the main problems, challenges, and research opportunities in state of the art are discussed.

1.1.1. Organization of the minicourse

The remainder of the short course is organized as described below. Section 1.2 presents the fundamental concepts involved in the theme of this work. The Internet of Things (IoT), Cloud Computing, and Fog Computing concepts are presented. In addition, the main threats present in IoT environments are also discussed, and concepts related to Intrusion Detection Systems (IDS) are introduced. Next, Section 1.3 discusses applying machine learning in the context of intrusion detection. Some classification techniques that can be used for intrusion analysis and detection are discussed. In addition, some practical aspects of each technique are presented. This section also proposes to conduct simulation

experiments with the IoTID20 dataset to evaluate the techniques presented in an intrusion detection scenario with IoT traffic. In Section 1.4, state of the art is exposed, and the approaches proposed by the main related works are presented. Section 1.5 discusses some important aspects observed in state-of-the-art related to intrusion detection in an IoT/Fog/Cloud context. The objective is to instigate an initial reflection on this research topic's problems, challenges, and open questions. Finally, Section 1.6 concludes the mini-course and presents final considerations and direction for future work.

1.2. Concepts and Technologies

In this section, concepts related to the theme of this work are addressed. Initially, the concepts of the Internet of Things (IoT), Cloud Computing, and Fog Computing, present in the environment chosen for this scenario, are contextualized. Section 1.2.4 discusses the main threats in IoT environments. Then, important aspects of the Intrusion Detection System (IDS) are introduced.

1.2.1. Internet of Things

The Internet of Things (IoT) has as its basic characteristic the pervasive presence of a wide variety of intelligent objects in people's daily lives, such as sensors, tags of Radio-Frequency IDentification (RFID), mobile phones, among others [Atzori et al. 2010]. The IoT connects physical devices to the Internet, enabling them to communicate and act intelligently.

From a conceptual point of view, IoT is based on three basic principles related to the characteristics of smart objects: being identifiable, communicable, and capable of interacting with the environment in which they are inserted [Miorandi et al. 2012]. IoT allows various physical objects to see, hear, feel, think, and communicate to share information and make decisions to perform certain tasks.

IoT applications can improve people's lives and how they live, work, learn, and have fun. For example, smart homes can provide residents with certain practicalities, such as automatic garage openings, automatic coffee preparation, climate control systems, etc.

IoT devices are small physical objects with limited processing and storage capabilities. Due to the large amount of data generated by these devices, there is a need for greater computational capacity. Furthermore, the number of devices connected to the Internet continues to grow. Cisco predicts that the number of interconnected devices on the planet could reach the 500 billion mark by 2025 [Camhi 2015]. Cloud computing can be a solution to solve these needs for greater processing capacity.

1.2.2. Cloud Computing

The National Institute of Standards and Technology (NIST) defines Cloud Computing as a model for enabling ubiquitous and convenient network access to a shared pool of configurable computing resources that can be quickly provisioned and released with minimal management effort or interaction between service providers [Mell et al. 2011].

According to the authors [Takabi et al. 2010], Cloud Computing is an important paradigm with the potential to significantly reduce costs through optimization and increased operational and economic efficiencies. They point out that this paradigm can significantly improve collaboration, agility, and scalability, enabling a truly global computing model over the Internet's infrastructure.

Cloud Computing has five essential characteristics: on-demand self-service, ubiquitous network access, pooling of resources, location independence, rapid elasticity, and measured service, all aimed at transparently using clouds. In the cloud, the provider's computing resources are pooled to serve multiple consumers using a multi-tenancy model. Different physical and virtual components are dynamically assigned and reallocated ac-

cording to consumer demand. Providers must provide rapid elasticity, allowing the consumer to increase or decrease resources and on-demand service so that the customer can unilaterally allocate resources dynamically [Takabi et al. 2010]. Cloud providers can handle large amounts of data and high processing rates.

The basic characteristics of cloud computing make it an important processing mechanism for IoT applications that capture large amounts of information. However, its use also has disadvantages, as this centralization of processing and storage resources implies a great separation between the physical IoT devices and the data centers of the cloud. This fact, which according to [Satyanarayanan 2015], results in the growth of average latency and jitter.

Then came Fog Computing, capable of solving the abovementioned problems for IoT applications. It extends the cloud closer to the user so that data access, processing, and storage tasks are performed by local resources such as routers, gateways, and switches. Therefore, the processing and storage of temporary data and the execution of local analyzes are carried out without long transmissions over the Internet. This way, fog computing doesn't suffer from high latency and jitter problems [Ni et al. 2018].

1.2.3. Fog Computing

Authors [Bonomi et al. 2012] define Fog Computing as a highly virtualized platform that provides computing, storage, and network services between IoT devices and cloud data centers. In addition, it is generally close to IoT devices at the edge of the network, as seen in Figure 1.1.

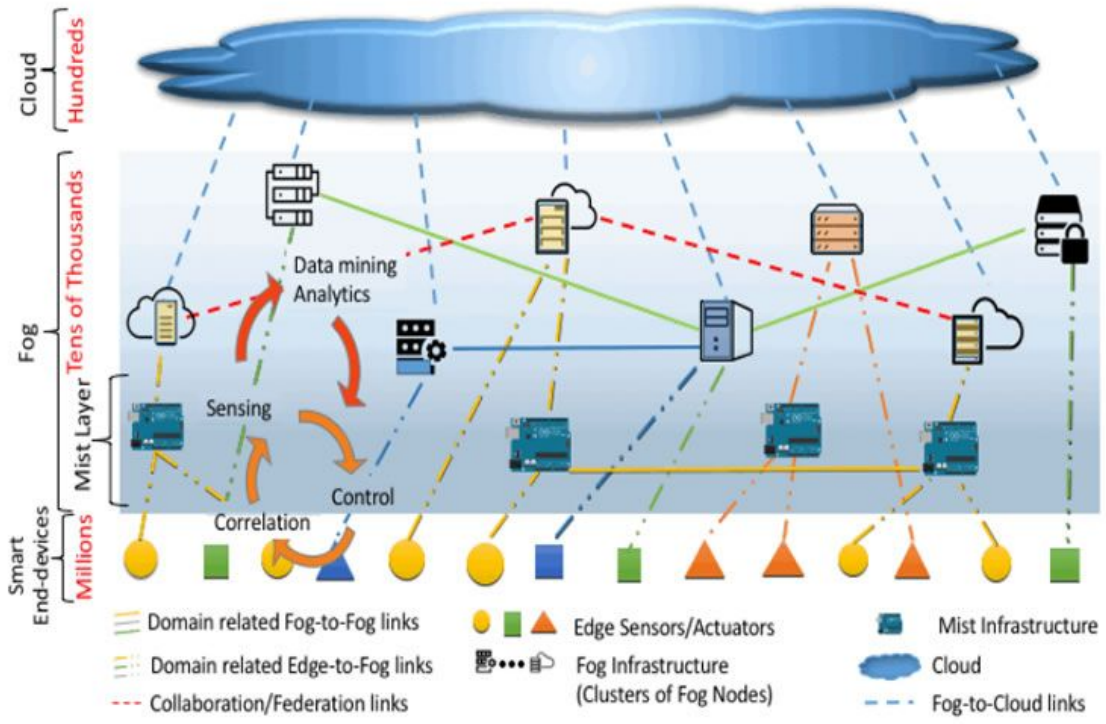

Figure 1.1. Illustration of existing layers in an environment based on the IoT-Fog-Cloud architecture [Iorga et al. 2018].

The main idea of this paradigm is to extend cloud computing closer to end devices to provide efficient data access, processing, and storage. Therefore, the hallmark of fog computing is the distribution of resources, communication services, processing, and storage close to users [Marín-Tordera et al. 2017].

Fog computing did not emerge to replace cloud computing for remote processing and storage but rather to complement it. Allowing the creation of a hierarchical infrastructure where local data is processed and stored by fog computing and permanent storage and global analysis are performed in the *datacenters* of the cloud [Ni et al. 2018].

As it is a recent paradigm, research into security and privacy issues is still at an early stage.

1.2.4. Threats in IoT and Fog Computing

This work considers the context of intelligent environments based on fog computing and IoT. Security in these environments is paramount, as IoT devices are often embedded in people's daily lives and handle sensitive information. In addition, some systems perform monitoring and critical actions, which need uninterrupted operation.

IoT and fog solutions comprise various technologies, services, and standards, each with security and privacy requirements [Zarpelão et al. 2017]. The IoT paradigm presents several security vulnerabilities that communication networks, cloud services, and the Internet have [Zarpelão et al. 2017]. However, traditional security tools have difficulties to be applied directly in this context due to three fundamental aspects: the limited computational power of IoT components, the high number of interconnected devices, and the sharing of data between objects and users [Sicari et al. 2015]. Furthermore, the rapid expansion of IoT solutions has left these networks vulnerable to security and privacy risks. Authors [Kolias et al. 2016] discovered several security vulnerabilities by creating IoT use cases using popular commercial products and services. Fog computing has emerged to provide greater computing resources for the IoT and low latency and compute-intensive use. These facts make it a great place to deploy IoT security applications like intrusion detectors [Nguyen et al. 2019]. However, security research on fog computing and IoT applications is still at an early stage [Ni et al. 2018]. This fact, combined with the great damage that attacks in this environment can cause, generates the need to concentrate efforts in this area. The authors [Garcia-Morchon et al. 2013] organized threats to the security of IoT environments into the following categories: device cloning, malicious replacement of devices, replacement of firmware, extraction of security parameters, interception, Man-In-The-Middle (MITM), routing attack, Denial of Service (DoS) and Distributed Denial of Service (DDoS). Threats related to cloning, replacement, and extraction usually occur during device manufacture, installation, maintenance, and updating [Zarpelão et al. 2017].

[Kolias et al. 2016] highlight the main attacks present in IoT: DoS, DDoS, MITM, routing attacks, and conventional attacks. Security threats related to conventional technologies that are part of the IoT environment may also apply, for example, insecure connections, malicious code injection, probing, interception, fabrication, and modification of messages [Muhammad et al. 2015].

As illustrated in Figure 1.2, these environments are subject to attacks from external sources, from the Internet, and internal attacks, by malicious devices in the IoT network. Malicious entities outside the network may attempt to gain privileged access to IoT devices to control them [Muhammad et al. 2015]. Through this access, it is possible to carry out botnet attacks, where compromised IoT devices can be used as bots or zombies to perform various malicious tasks [Aversano et al. 2021]. Furthermore, spoofing attacks involve impersonating legitimate devices, exploiting their identities to gain access to the IoT network, and then launching other types of malicious actions [Aversano et al. 2021], such as stealing confidential information handled in the IoT network. DoS attacks are quite common and aim to affect the victim's availability. This can be done by flooding with a large volume of requests or depleting resources such as memory and computing power. In the context of IoT, the device can be part of the network under threat or be used as a zombie to launch a DDoS on another network. Probing attacks, where the attacker scans a network to gather information and discover vulnerabilities, are common. This attack usually collects information from the target before launching another, more severe type of attack [Arshad et al. 2019].

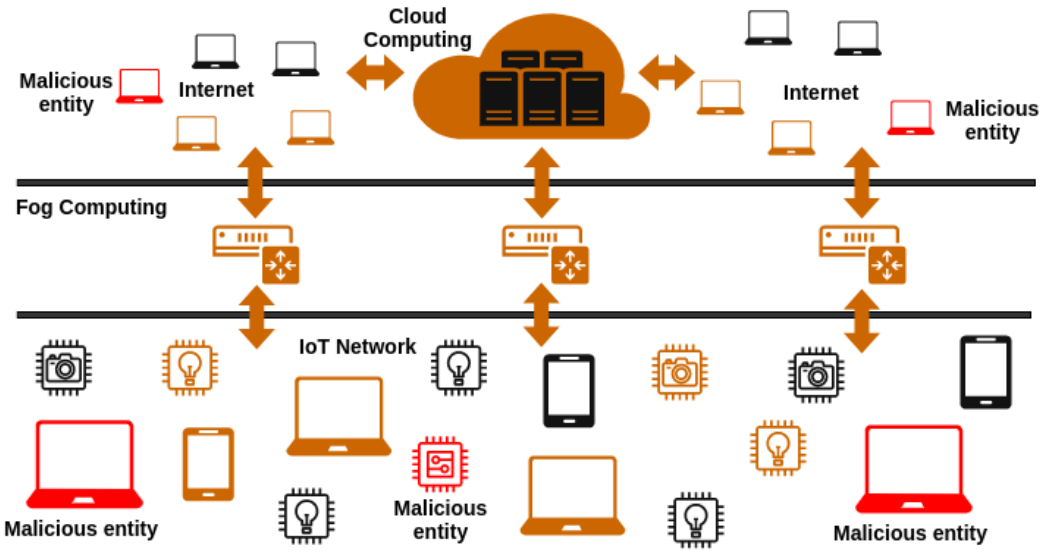

Figure 1.2. Illustration of potential threats in an IoT and fog computing environment.

In the case of an insider attack, the attacker is presumed to be a malicious entity that was successfully authenticated in the fog or a legitimate IoT device that has become malicious over time. It can carry out various attacks, including denial of service, by sending a high data packet rate in the fog. This allows overloading the devices and the upper layers, as illustrated in Figure 1.2, impairing or even causing the interruption of services provided by the systems, services that in many cases are extremely important. Therefore, these actions can damage the entire IoT system, and external users who access services and information generated by the IoT solution can be harmed.

Based on the classification presented by the [Zarpelão et al. 2017] authors, the main attacks found in the fog and IoT environment are presented below.

- Denial of Service (DoS): Denial of Service attacks aim to affect the availability of the victim. This can be done by flooding with a huge volume of requests or depleting resources like memory and computational power [Zarpelão et al. 2017]. Both an IoT node and the fog can fall victim to this attack.

- Distributed Denial of Service (DDoS): Distributed denial of service attacks have the same objective as a DoS. However, they are executed by a set of hosts, whereas in a common DoS, a single host is the attacker [Yan et al. 2016]. Furthermore, in the context of the Internet of Things, the IoT node can be part of the network under threat or be used as a zombie to launch a DDoS on another network.

- Man-In-The-Middle (MITM): a man-in-the-middle attack is performed when an attacker interferes with the communication between an entity A and an entity B, without A and B realizing it [Zarpelão et al. 2017]. Authors [Navas et al. 2018] demonstrate the risks of MITM attacks on IoT networks caused by malicious insider devices.

- Routing attacks: Routing attacks consist of spoofing, modifying network routing information to create loops, attracting or rejecting traffic, extending or shortening routes, etc. Other possible routing attacks include sinkhole attack, selective forwarding, wormhole attack, and sybil attack [Zarpelão et al. 2017].

- Conventional attacks: Security threats related to conventional technologies that are part of the IoT environment can also apply to IoT systems, for example, insecure connections, malicious code injection, interception, probing, fabrication, and modification of messages [Muhammad et al. 2015].

In addition to the existing threats related to computer networks, IoT needs to deal with the resource constraints that its devices have. Intrusion detection and prevention systems can be used to secure IoT networks. The basic concepts related to these systems are presented below.

1.2.5. Intrusion Detection

An intrusion can be defined as a set of actions to overcome an application's defense barriers to compromise the integrity, confidentiality, and availability of computational resources [Heady et al. 1990]. Intrusion Detection Systems (IDS) are intended to recognize intrusive actions and behavior to alert administrators or automatically execute countermeasures [Bace and Mell 2001].

Intrusion detection research efforts have been conducted since 1980. Around that time, [Anderson 1980] presented a threat model and security monitoring system based on detecting anomalies in user behavior. IDSs are inserted as the last line of defense within a computational architecture, making them of great importance, making it possible to infer the legitimacy of actions taken and having a proactive behavior in attack situations [Patel et al. 2010]. The structure of the IDSs can vary in relation to the way it is implemented, the frequency of operation, the data they analyze, or the analyzes carried out on them [Campello and Weber 2001].

IDSs can be classified according to the detection methods employed. Thus, they can be classified into analysis by signature or behavior, also known as analysis by anomaly.

In signature detection, monitored actions are compared with predefined intrusive events, normally stored in a database. These previously known patterns are called signatures. Signature detection allows quick detection and reduces the occurrence of false alarms. However, it has the limitation of detecting only known attacks, that is, only attacks with a signature known by the IDS [Northcutt et al. 2001]. Most commercial antivirus systems use this strategy [Bace and Mell 2001].

Anomaly detection assumes that any abnormal activity is necessarily an intrusion, and any activity that does not fit the defined normal behavior models is considered an attack. The great advantage of the anomaly detection technique is that it allows the detection of new attacks and/or variations of already known ones since it is not necessary to know about them previously. However, this technique is more likely to suffer from problems related to false positives [Boukerche et al. 2007]. This strategy is usually modeled using Machine Learning techniques. Section 1.3 presents more details about these techniques.

In addition, some works consider a branch of analysis by behavior called analysis by specification [Mitchell and Chen 2014]. This type of solution employs rules and thresholds that define the expected default behavior for monitored components. It is similar to anomaly detection, and both detect intrusions when network behavior deviates from specified. The main difference is that in specification-based analysis, a human expert sets the rules [Mitchell and Chen 2014, Zarpelão et al. 2017]. This type of analysis's major drawback is the specificity and domain knowledge required to specify benign behavior.

IDSs can analyze data from multiple sources and can be deployed in different locations. These data are generally related to how the approach is implemented. There are two main categories of implementation related to capturing information. The Host-Based Intrusion Detection System (HIDS) seeks to analyze the information captured from the very host where they are deployed, and Network-Based Intrusion Detection System (NIDS) analyzes traffic captured from the monitored network [Zarpelão et al. 2017]. Furthermore, in the context of IoT, approaches can be deployed at different levels: on IoT devices themselves, on devices in the fog, or in the cloud.

In a host-based IDS, all components, from event collection to classification, are located on the same host. The event monitoring and analysis mechanisms only use information from the host itself. Events can originate from system logs and data about users, services, and processes. This approach enables network independence and the detection of insider attacks. However, host-based solutions employed on IoT nodes may suffer from memory constraints. Those employed in fog devices allow the detection of attacks against the device itself but may have difficulty dealing with attacks on the network and IoT devices.

On the other hand, NIDS approaches are implemented in a device capable of capturing the network traffic intended to be monitored. Events and activities are obtained by capturing network traffic in promiscuous mode. These IDSs typically monitor a network made up of multiple devices. Sensors can also be used to capture information

in a distributed manner at various points in the network. One of this approach's difficulties is determining the best places to position the information capture sensors. The analysis method, in these approaches was generally included in the fog devices. This strategy allows the detection of external attacks and is more independent of the platform [Mukherjee et al. 1994]. Fog computing is one of the most promising alternatives for implementing IoT network monitoring approaches. Furthermore, it is interesting to divide the detection tasks along the complete architecture, considering IoT devices, fog, and cloud.

In addition to detecting intrusions, it is very important to have mechanisms to execute countermeasure actions, with the aim of blocking and preventing the intrusion from succeeding. Among the existing actions are issuing alerts to the network manager. Issuing just one alert does not configure a prevention action, as it only makes the manager aware that an intrusion has occurred, but does not prevent it. Issuing an alert is considered a passive post-detection. Another class of post-detection approaches is the active one, where the actions taken aim to stop an attack in progress and then block the attacker's access [Bace and Mell 2001]. IDSs that have active countermeasures are known as Intrusion Prevention Systems (IPS) [Birkinshaw et al. 2019].

In the following sections, other concepts involved in the context of this work are presented. Section 1.3 initially presents the basic concepts of ML and a brief description of its applicability for intrusion detection. Next, several classification techniques that are employed in behavioral detection approaches are presented.

1.3. Machine Learning techniques employed for intrusion detection

In [Russell and Norvig 2010], several definitions are presented for the term Artificial Intelligence (AI), which, in general, point to this as the ability to make machines reproduce intelligent activities and cognitive abilities found in humans. When we analyze more emphatically the currently proposed solutions in the area of computational security, there is increasingly stronger research and application of machine learning methods for improvements in the intrusion detection process.

According to [Goodfellow et al. 2016], ML is the ability of a given technique to acquire its knowledge, extracting information from raw data and representing it through some kind of mathematical model. ML has several sub-areas, including classification. A classification task consists of classifying data and objects into certain classes in an automated way.

It is observed that this ability of the machine learning classification methods fits perfectly with the context of intrusion detection since the detection approaches have the task of analyzing the information captured from the network or hosts, verifying the occurrence of abnormal behaviors, and performing the classification of information in benign or intrusive. In addition, there is also a need to classify attacks into types or categories. Therefore, classification techniques are great for composing anomaly-based detection approaches.

Machine learning methods usually need to train to acquire knowledge and generate a model with added knowledge. Next, the two most common types of learning employed in solutions found in the state of the art of intrusion detection are presented.

The main characteristic of approaches based on supervised learning is the existence of labels in the subset of training data. This type of learning reflects an algorithm's ability to generalize knowledge from available data with target or labeled cases so that the algorithm can be used to predict new unlabeled cases [Berry et al. 2019]. Thus, the method training process uses this prior knowledge to train and generate the classification models. After training, the methods can classify new data. This approach's difficulty lies in need for labeled data for training the models [Russell and Norvig 2009].

Unsupervised learning refers to grouping data into unlabeled data using automated methods or algorithms. In this situation, algorithms need to understand the underlying relationships or features of the available data and group cases with similar features or characteristics [Berry et al. 2019].

In this section, several Machine Learning (ML) techniques used in intrusion detection in fog computing and IoT environments are presented and discussed. They are often employed in behavior-based detection strategies. The main focus of this short course will be methods based on supervised learning, as they are the most used in this context of intrusion detection. Next, the K-Nearest Neighbors (KNN) method is presented first.

1.3.1. K-Nearest Neighbor (KNN)

The k-Nearest Neighbors (kNN) algorithm is one of the most basic instance-based learning methods. It assumes that all examples correspond to points in an n-dimensional plane R^n, where n is the number of attributes used to represent them. Despite its simple oper-

ation, KNN generally has a very low error rate. This method uses a distance function to determine one instance's proximity to another [Mitchell 1997].

When numerical attributes describe the data set, distance measures are used to calculate the similarity so that the smallest distance corresponds to the greatest similarity. The Euclidean Distance [Mitchell 1997] stands out among the commonly applied measures.

The Euclidean distance is calculated as the square root of the sum of the squared differences between points of instance p in relation to points of instance q, as can be seen in Equation 1. Since p_i and q_i, for $i = 1, 2, ..., n$, are the attributes n that describe the instances p and q, respectively.

$$Euclidean\ Distance\ (p,q) = \sqrt{(p_1 - q_1)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{1}^{n}(p_i - q_i)^2} \quad (1)$$

The K-Nearest Neighbor (KNN) algorithm identifies the closest k neighbors to the new data point and classifies it according to the nearest neighbors [Mitchell 1997]. If $k > 1$, the classes of the closest k examples are considered to carry out the classification. In this case, the most common approach is to assign the new instance to the majority class present in the set of the closest k examples.

Figure 1.3 presents a basic implementation of the KNN algorithm for data classification. As can be seen, the method has some parameters that influence the classification process. The main one is *n_neighbors*, corresponding to the number of neighbors considered for the classification process.

```
from sklearn.neighbors import KNeighborsClassifier

method = KNeighborsClassifier(n_neighbors=1, weights='uniform', algorithm='kd_tree')

method.fit(training_data_samples, training_data_labels)

predictions = method.predict(testing_data_samples)
```

Figure 1.3. Example application of the KNN algorithm in Python using the scikit-learn library.

Furthermore, the parameter *weights* indicates the influence of *n_neighbors* on the ranking. In the case of "uniform" weights, all neighbors are weighted equally. On the other hand, with "distance" weights, among the k neighbors, the closest ones will have more influence than the farther neighbors. The *algorithm* parameter indicates the technique used to store the training data, usually transformed into a fast indexing structure, such as Ball Tree ("ball_tree") or KD Tree ("kd_tree"). A KD Tree is a binary tree where each node is a k-dimensional point. The standard metric used in this library to calculate the similarities between the data is the Minkowski distance. It is a metric in a normalized vector space that can be considered a generalization of the Euclidean distance. As can be

seen in Figure 1.3, the algorithm is trained using the *fit()* method, where training data and labels are passed as parameters. The *predict()* method is used to carry out the classification, and data without labels is provided. The method will then generate class predictions for this data as a result.

The KNN algorithm is commonly used in intrusion detection approaches due to its low error rate [Illy et al. 2019]. An existing disadvantage in KNN is the computational cost, which can become high because it is necessary to compare the new instances with all the instances stored in the example base.

1.3.2. Artificial Neural Networks (ANN)

The brain has densely interconnected neurons forming a highly complex structure. Inspired by it, the Artificial Neural Networks (ANN) was proposed [Haykin 2001]. The artificial neuron is a logical mathematical structure that aims to simulate the biological neuron's shape, behavior, and functions. Inputs replace the dendrites, and the connections of these inputs with the artificial cell body are known as weights, which simulate synapses. The summation function processes the stimuli received by the inputs. The firing threshold of the biological neuron is simulated by the activation function in the artificial neuron [Chua and Yang 1988]. According to [Dalton and Deshmane 1991], synaptic weights play an important role in artificial neurons. The purpose of the weights is to weigh the influence of input signals on postsynaptic neurons. Positive weights tend to increase a neuron's activation level, consisting of an excitatory connection. Negative weights, on the other hand, tend to decrease the level of activation, which are called inhibitory connections.

Figure 1.4 shows a simplified model of an artificial neuron. Where the neuron k receives an input x $(x_1, x_2, ..., x_n)$ that enters through the synapse j. Each signal x_j from the input x that enters the synapse j is multiplied by a weight w_{kj}. The result of this process passes through an adder that adds the input signals weighted by the weights of the respective synapses with an external *bias* (b_k). Concerning a given synaptic weight w_{kj}, the first index (k) refers to the neuron in question, and the second index (j) refers to the input of the synapse to which the weight is related to [Haykin 2001]. The previously mentioned bias b_k has the role of increasing or decreasing the value generated by the adder before passing it on to an activation function, which will then transform the output into a closed interval, usually between $[0, 1]$ or $[-1, 1]$ to be passed on to other neurons [Haykin 2001].

The activation function, which in Figure 1.4 is represented by $\varphi(.)$ is extremely important in the neural model, as it defines what the neuron's output will be according to the received input [Haykin 2001]. Therefore, the activation function $\varphi(v)$ defines the neuron's output according to the result of the sum (v) of the weighted inputs. Several activation functions are used in neural networks, including Binary Step, Logistic Sigmoid, Hyperbolic Tangent, ReLU, and Softmax.

The Binary Step activation function defines the output of the summation result in v. In this model, if the result of the adder of the neuron in question is greater than or equal to 0, that is, it is positive, the output of the neuron assumes a value of 1, and if the result of the adder is negative, the output value of the neuron will be 0.

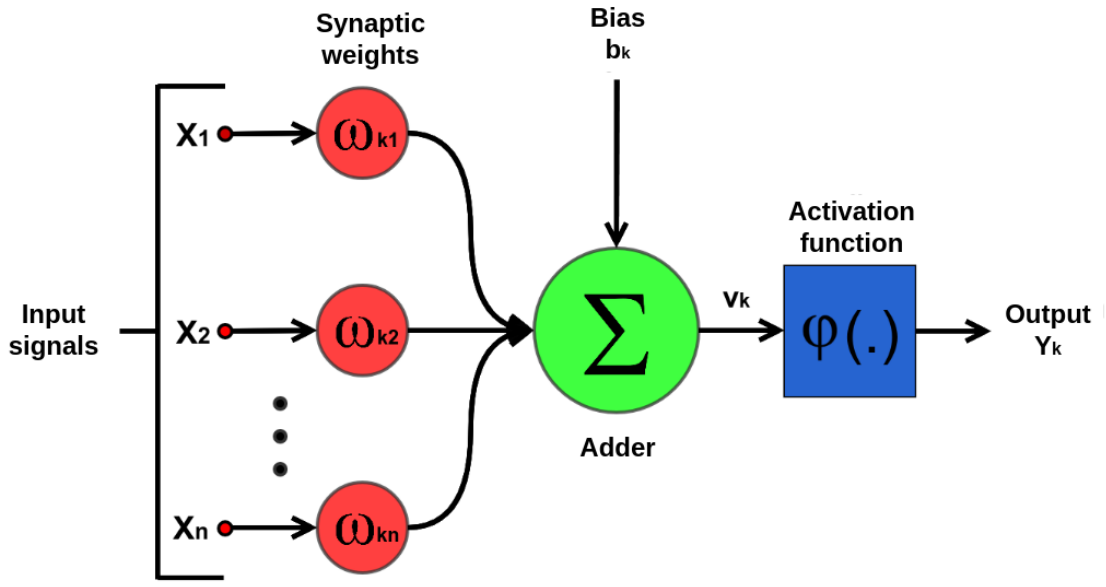

Figure 1.4. Simplified model of an Artificial Neuron. Adapted from [Haykin 2001].

The sigmoid activation function has an "S" shaped graph and assumes a continuous range of values between 0 and 1. It is one of ANN's most common activation functions and exhibits a good balance between linear and non-linear behavior. An example of a sigmoid function is the Logistic function, where a is the slope parameter of the sigmoid function. When this parameter tends to infinity, the function approaches the threshold function. While the threshold function assumes the value of 0 or 1, the logistic function assumes a continuous range of values between 0 and 1.

The hyperbolic tangent function is similar to the logistic sigmoid, but its continuous values range from -1 to 1. Allowing an activation function to assume negative values provides analytical benefits and advantages during the training phase [Haykin 2001].

Softmax generalizes the sigmoid function for non-binary cases. It is not usually applied to the hidden neural network layers but to the multiclass classification problems' output layer. The softmax function transforms the outputs for each class to values between 0 and 1 and divides them by the sum of the outputs. It is the probability that the input is in a given class.

Finally, ReLU is an abbreviation for Rectified Linear Unit (ReLU). It returns 0 for all negative values and the value itself for positive values. Thus, if the input is negative, the neuron will not be activated. This means that only a few neurons are activated simultaneously, making the network sparse and efficient. Therefore, it is a computationally light function widely used in hidden layers of neural networks [Goodfellow et al. 2016].

Artificial neural networks are composed of a large number of artificial neurons organized in the input layer, hidden layers, and output layer [Haykin 2001]. How neurons are arranged in an ANN is directly related to the learning algorithm. For this work, we will approach the class of networks Multilayer Perceptron (MLP) feedforward [Russell and Norvig 2010]. In this way, neurons in one layer are connected to neurons

in the next layer through weighted links. There are no recursive bindings present in this network topology. This model does not have feedback loops between neurons, so the flow of the synaptic process occurs from the input layer toward the output layer. The architecture displayed in Figure 1.5 illustrates a common basic structure of MLP neural networks. However, there may be several variations in the design of the hidden layers, both in terms of the number of layers and the number of neurons in each. These intermediate or hidden layers' design is crucial in defining a neural network, especially Deep Neural Networks (DNN). DNNs are deep networks, with more than one hidden layer [Goodfellow et al. 2016].

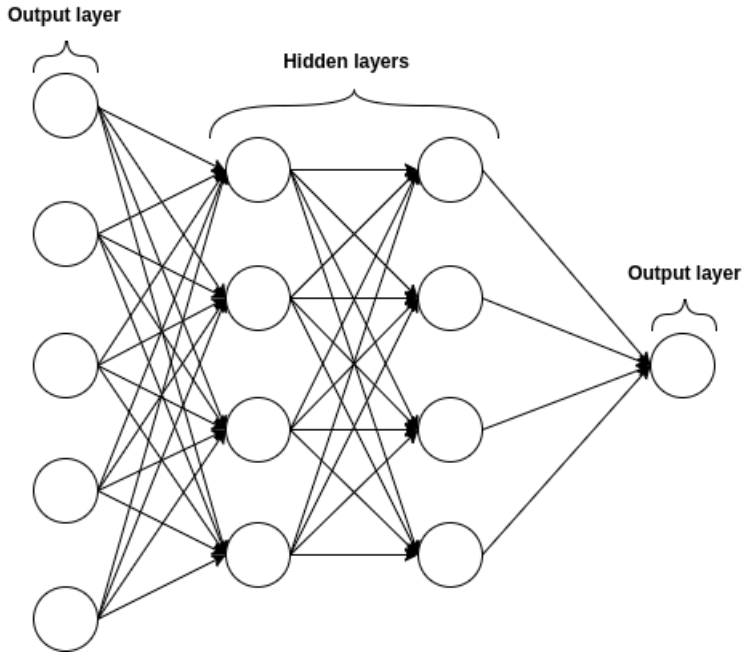

Figure 1.5. Simplified architecture of a deep feedforward neural network with two hidden layers.

An essential characteristic of an ANN is its ability to learn from its environment and improve its performance through training. An ANN learns more about its environment through an iterative process of adjusting its synaptic weights and bias levels, and this process is defined as training. The neural model is generated through the supervised training process, where the link weights are updated in several iterations based on the estimated error. ANN becomes more knowledgeable of its environment after each iteration of the training process [Haykin 2001]. ANN knowledge is represented through synaptic weights, forming a compact and distributed representation, thus providing generalization capabilities and adaptability to the neural network. ANN manage to achieve great ranking performances. However, they may suffer from instabilities caused by noise and variance in training. This instability means small changes to the training data used to build the model can result in very different models [Cunningham et al. 2000].

Training of MLP networks is usually performed using the backpropagation algorithm. The training takes place by propagating the data from the input layer to the output, passing through each of the hidden layers, and at that moment, the weights remain un-

changed. Afterward, based on the error calculated using the expected result (supervised learning) and the output value of the last layer, the weights are adjusted, and a new training iteration is performed. Training is considered completed when the error is small enough. From this, the network starts operating only in the forward direction for classifying new examples.

Figure 1.6 shows an implementation example of a basic DNN for classification, which can be used for intrusion detection. In this example, Keras and Tensorflow libraries are used. Keras *Sequential*() handles the ordering or sequencing of layers within a model. It makes the layers associated with neural networks work like a model that receives only one input as a feed and expects an output. The *add*() method adds layers to an already created layer stack. The *Dense*() layer is the regular deeply connected neural network layer and is the most common and frequently used. The *fit*() method is used to train the model, and the *predict*() method returns the values generated by the output neurons. The *np.argmax*() method obtains the neuron with the highest output value.

```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation

method = Sequential()
method.add(Dense(10, input_dim=79, activation="relu"))
method.add(Dense(10, activation="relu"))
method.add(Dense(5, activation="softmax"))

print(method.summary())

method.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

method.fit(training_data_samples,training_data_labels, epochs=10, verbose=2)

predictions = np.argmax(method.predict(data_samples[test]),axis=1)
```

Figure 1.6. DNN implementation example using Keras and Tensorflow library.

1.3.3. Ensemble Learning

Ensemble Learning (EL) is the field of study of machine learning that works with Ensemble methods, which combine the decisions of various classification models to improve overall performance. The predictions from the various models are combined in some way to generate the final prediction [Breiman 1996].

Individual classifiers may experience instability. There is no guarantee that a classifier will always perform at its best in all situations. However, with ensemble learning, a better classification performance than any individual classifier can be achieved [Traganitis et al. 2018].

The main idea of ensemble learning is that combining several models decreases variation, especially in the case of unstable classifiers. In this way, it is possible to produce a more reliable classification than a single model [Breiman 1996].

1.3.3.1. Bagging

One of the main strategies for creating ensemble methods is Bagging (Bootstrap Aggregating). It generates multiple models of a classifier and uses them to obtain an aggregated classifier. Random samples of the training dataset are created for each model. Several subsets of the training dataset are created, and each model is trained with a subset. Finally, the results of these various models are combined using average or majority voting. Tests on real and simulated datasets using classification trees show that Bagging can provide substantial gains in accuracy [Breiman 1996].

Figure 1.7 presents an implementation example of a bagging method. The *n_estimator* parameter indicates the number of classifiers that will compose the method and the *estimator* indicates which is the base classifier.

```
from sklearn.ensemble import BaggingClassifier

method = BaggingClassifier(estimator=classifier,n_estimators=100)

method.fit(training_data_samples, training_data_labels)

predictions = method.predict(testing_data_samples)
```

Figure 1.7. Ensemble bagging application example in Python using the scikit-learn library.

Two specific ensemble methods for decision trees are presented below: the Random Forest and Extra Tree algorithms. These techniques create a diverse set of decision tree classifiers by introducing randomness in constructing the classifier.

1.3.3.2. Random Forest (RF)

Random Forest (RF) is an ensemble learning method based on Decision Tree (DT) created to reduce Overfitting. Decision Tree consists of a supervised machine learning algorithm based on the idea of recursively dividing a more complex problem into simpler problems. The input data are divided into homogeneous groups where each division performed represents a node of the tree where the data are separated according to a division criterion until reaching indivisible points. DTs have a tree-like structure composed of nodes. These nodes can be divided into a root node, a set of intermediate nodes, and a set of leaf nodes [Breiman et al. 1984]. The root node corresponds to the first division specifying how the data should be divided into separate parts. Successive intermediate nodes divide the data into smaller partitions until no further partitioning is needed. In this way, the leaf nodes of the structure represent the final partitions [Rokach 2016]. DTs classify using a hierarchical set of feature decisions. The decisions made in the internal nodes are the division criteria.

The basic decision tree induction algorithm builds decision trees recursively on a divide-and-conquer basis, starting from the top down. Each iteration seeks the attribute capable of best dividing the dataset. A good split in a decision tree corresponds to choos-

ing the attribute with the maximum separation power. In other words, the purpose of each node is to create child nodes dominated by a single class. The most suitable attribute is selected according to specific division criteria. Attributes are evaluated according to the division criterion, with the best attribute selected. The process is recursive, so each node further subdivides the training set into smaller subsets after selecting an appropriate split. For numeric attributes, there are many possible cut-off points. The induction algorithm looks for the best cut-off point by evaluating the split criterion at each possible cut-off point [Rokach 2016]. When the node satisfies the stopping rules, for example, because all instances of the current partition belong to the same class or no future split attributes can be determined, the DT terminates the splitting process, and the node is labeled [Rokach 2016]. One of the significant challenges of decision tree algorithms is finding the attribute that best divides the data into its corresponding classes. The main metrics used for this are the Gini Index and information gain based on entropy.

The Gini coefficient measures how well a given attribute separates the classes contained in a node. Possible values for the Gini index vary between 0 and 1, where 0 expresses the purity of the classification. All elements belong to a certain class, or only one class exists. Furthermore, 1 indicates a unequal distribution [Sundhari 2011]. The Gini index is determined by subtracting the sum of squared probabilities for each class from 1. It is expressed mathematically in Equation 2 [Breiman et al. 1984]. Where P_i denotes the probability that an element is classified into a class. When building the decision tree, resources with the lowest values of the Gini Index are chosen [Sundhari 2011].

$$GiniIndex = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{2}$$

Another way to measure the quality of an attribute is to evaluate its degree of association with the class through the Information Gain measure. It evaluates the degree of association of attributes with the class to find the values with the highest degree of use and importance by calculating the entropy reduction. The greater the entropy, the greater the degree of impurity. The information gain indicates the entropy reduction. Thus, the attributes with the highest information gain will be the most useful for detection.

The measure of information gain is based on the concept of entropy. Entropy is a measure of the impurity and inhomogeneity of an attribute. The formula presented in Equation 3 corresponds to the entropy calculation for an attribute A, whose domain is $(a_1, a_2, ..., a_k)$, with $k >= 1$. The values p_i, with $1 <= i <= k$, correspond to the ratio between the number of instances of the base in which the value a_i occurs for the attribute A and the total number of instances.

$$Entropia(A) = \sum_{i=1}^{m} p_i \log_2(p_i) \tag{3}$$

Decision trees can have problems related to overfitting, which can degrade their predictive power when applied to new data [Breiman 2001]. In addition, they are considered models that can be unstable, where small variations in the training data can result in completely different trees. This can be avoided by training several different trees and

aggregating their predictions. Below are several methods based on the aggregation of ML models. This strategy is known as Ensemble Learning (EL) and seeks to generate methods with lower variance and more reliability.

RF builds a "forest" with a large number of uncorrelated decision trees and combines the results yielding the final classification results. Provides an additional layer of randomness over Bagging. In addition to building each tree using a different bootstrap sample of the data, RF changes how classification trees are built. Unlike standard DT, where each node is split using the best split among all features, in RF, each node is split using the best among a subset of features chosen randomly on that node [Breiman 2001]. It can get very good performance compared to many other classifiers like SVM and neural networks. Furthermore, it is robust against overfitting [Breiman 2001].

One of the main characteristics of RF is using a degree of randomness in selecting attributes to be considered for the division. Unlike DT, which applies impurity metrics across the entire set of attributes to find the best, RF applies these metrics only to a randomly selected subset of candidate attributes. Furthermore, it uses only a subset of the training data, with replacement, to construct each structure tree. The algorithm searches for the attributes that generate the best separability in each tree node. It randomly selects a set of candidate attributes and applies the measures to find the best cutoff point for each attribute and the best attribute among the candidates. This process ensures that each tree generates a different model. After RF training, the structure can perform the classification of new data. Each generated tree will classify the record, and its results will be combined through average or majority voting.

The RF method has several parameters, as shown in Figure 1.8, which presents a basic implementation of using RF for classification.

```
from sklearn.ensemble import RandomForestClassifier

method = RandomForestClassifier(n_estimators=100, max_features="sqrt",
                          criterion="gini", max_depth=100,
                          min_samples_split=2, min_samples_leaf=1)

method.fit(training_data_samples, training_data_labels)

predictions = method.predict(testing_data_samples)
```

Figure 1.8. Random Forest application example in Python using the scikit-learn library.

In RF, it is possible to define the parameters related to the internal structures of the decision trees that make up the RF structure. One of the parameters is the *criterion* that defines the metric used to choose the best attributes, where the metrics mentioned above can be used: Gini Index ("gini") and Entropy ("entropy"). The parameter *max_depth* indicates the maximum allowed depth of the tree. It is used to control this growth because normally, they can grow until all the leaves are pure or until all the leaves contain less than *min_samples_split* samples. However, this can lead to extremely long and costly trees. The parameter *min_samples_split* indicates the minimum size of the training set to split a node, and the (*min_samples_leaf*) indicates the minimum number of samples needed

to form a leaf node. A split point in any depth will only be considered if you leave at least *min_samples_leaf* samples in each of the left and right branches.

Furthermore, there are some additional parameters specific to the ensemble approach, one of which is *n_estimators*, which corresponds to the number of trees that will be created in the RF structure. Another important parameter is *max_features*, which indicates the number of randomly selected features in each node, where *max_features* is responsible for the intensity of the feature selection procedure and *n_estimators* the strength of variance reduction of the aggregation of the ensemble model [Geurts et al. 2006]. The RF is trained through the *fit()* method and performs the classification with the *predict()* method, as can be seen in Figure 1.8.

1.3.3.3. Extra Tree (ET)

Extra Tree (ET) [Geurts et al. 2006] classifiers are important tools in classification tasks. Like the RF, the Extra Tree consists of an ensemble method aggregating the results of several uncorrelated DTs accumulated in a "forest" to produce the classification results.

ET focuses on heavily randomizing the choice of attributes and the cutoff point while splitting a node in the tree. Therefore, a random sample of resources from the resource pool is selected at each intermediate node. Each decision tree must select the best feature to split the data based on some mathematical criteria, usually the Gini index. In the extreme case, it builds totally random trees whose structures are independent of the learning sample output values [Geurts et al. 2006]. The prediction trees are aggregated to produce the final prediction by majority vote in classification problems and arithmetic means in regression problems [Geurts et al. 2006]. It is very similar in operation to RF and varies mainly in the way of building the DTs inside the forest. In ET, randomness goes a step further in how divisions are calculated. Cutpoints are randomly drawn for each candidate attribute, and the best of these generated cutpoints is randomly chosen as the splitting rule.

```
from sklearn.ensemble import ExtraTreesClassifier

method = ExtraTreesClassifier(n_estimators=100, max_features="sqrt",
                          criterion="gini", max_depth=100,
                          min_samples_split=2, min_samples_leaf=1)

method.fit(training_data_samples, training_data_labels)

predictions = method.predict(testing_data_samples)
```

Figure 1.9. Example application of ExtraTree in Python using the scikit-learn library.

The rationale behind the method is that precise slicing and attribute randomization combined with ensemble mean should reduce variance more strongly than the weaker randomization schemes used by other methods. Using original and complete training data rather than bootstrap replicates is motivated to minimize bias [Verma and Ranga 2020]. In addition to precision, the main strength of the resulting algorithm is computational

efficiency, because, given the simplicity of the node split procedure, the constant factor can be much less than in other ensemble methods that locally optimize the cut points [Geurts et al. 2006]. The ET method also allows the definition of parameters of the internal decision trees that compose the structure. Furthermore, as it is a very similar technique to RF, it presents similar parameters, as seen in Figure 1.9.

1.3.3.4. Boosting

Boosting is another ensemble technique. It produces a highly accurate classifier by combining several "weak" models, each of which may not be good for the whole data set but is good for part of the data set so that the performance of these classifiers is improved [Schapire 1990]. Like bagging, boosting trains each model using a different training set. It is an iterative approach that adjusts the weight of an observation based on the last ranking. The performance of previously generated models influences each generated model. The boosting strategy is to focus on poorly classified examples. Each new model is created to classify well the examples poorly classified by previous models [Meir and Rätsch 2003]. Boosting generally decreases bias error and creates strong predictive models.

Decision trees are susceptible to overfitting, and to solve this problem, the Gradient Boosting Decision Tree (GBDT) can be used. It consists of a machine learning algorithm with effective implementations like XGBoost. Although many engineering optimizations were adopted in these implementations, the efficiency and scalability still needed improvement when the resource dimension was high, and the data size was large. One of the main reasons is that they have to scan all data instances for each feature to estimate the information gain of all possible split points, which is very time-consuming. To solve this problem, the authors [Ke et al. 2017] proposed LightGBM, which is based on Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) techniques. The two techniques form the characteristics of the LightGBM algorithm, and they integrate to make the template work efficiently and provide an advantage over other frameworks. These techniques work around the limitations of the histogram-based algorithm primarily used in all GBDT structures.

```
from lightgbm import LGBMClassifier

method = LGBMClassifier(n_estimators=100)

method.fit(training_data_samples, training_data_labels)

predicoes = method.predict(testing_data_samples)
```

Figure 1.10. Ensemble LightGBM application example in Python using the scikit-learn library.

The GOSS technique excludes a significant proportion of data instances with small gradients and uses only the remainder to estimate the information gain. According to the information gain definition, those instances with larger gradients will contribute more to the information gain. Therefore, when downsampling the data instances, to maintain the

accuracy of the information gain estimate, one should better keep those instances with large gradients, for example, greater than a predefined threshold or between the upper percentiles, and randomly eliminate instances with small gradients. GOSS can get a very accurate estimate of the information gain with a much smaller data size [Ke et al. 2017].

EFB groups mutually exclusive features, which rarely assume non-zero values simultaneously, to reduce the number of features. Finding the optimal grouping of unique features is NP-hard, but according to [Ke et al. 2017] authors, a greedy algorithm can achieve a good approximation ratio and effectively reduce the number of features without greatly impairing the accuracy of the split point determination. Figure 1.10 presents the implementation with the LightGBM technique in an example in Python.

1.3.3.5. Voting

Combining conceptually different machine learning classifiers and using a voting mechanism to generate the final classification is also possible. In this scheme, all classifiers are trained with the complete dataset, and their predictions are combined through voting.

The types of voting that can be used are hard and soft. In hard voting, each classification technique votes for a class, and the class that obtains the most votes is the final classification. The other strategy consists of a combiner based on the maximum sum of prediction probabilities. In a simplified way, each classification technique provides a probability value that the instance belongs to a given class. The predictions are then summed, and the class with the highest sum of probabilities is defined as the final classification.

Figure 1.11 presents an example of implementing an ensemble voting strategy. In the case presented, 'hard' voting is used, however, the voting parameter (*voting*) can also receive the value 'soft'.

```
from sklearn.ensemble import VotingClassifier

method = VotingClassifier(estimators= [('name1', classifier1), ('name2', classifier2)],
                          voting='soft', weights=None)

method.fit(training_data_samples, training_data_labels)

predicoes = method.predict(testing_data_samples)
```

Figure 1.11. Ensemble voting application example in Python using the scikit-learn library.

The base classifiers used to compose the voting-based method are passed as a parameter *estimators* in sequential form following the notation: name of the classifier and the object of the classifier. Although the example depicts only two base classifiers, the approach can work with more. Finally, the parameter *weights* indicates weights to weight the votes of the base classifiers. By default, it assumes the value 'None' where all base classifiers have a vote with the same weight.

1.3.3.6. Stacking

The Stacking technique consists of an ensemble strategy that combines several machine learning algorithms through a metamodel. The various base-level algorithms are trained on a complete training dataset, and the metamodel is trained on the final results of the base-level models. The predictions made by the basic models serve as a resource for the metamodel. In this way, the metamodel is responsible for learning to combine the individual results of each base classifier into an overall final result [Kumar et al. 2021a].

Figure 1.12 shows an example of implementing the ensemble stacking strategy. The base classifiers used to compose the stacking-based method are passed as a parameter *estimators* in sequential form following the notation: name of the classifier and the object of the classifier. The $stack_method$ parameter indicates the method called for each base classifier. In the case of the example, each of the base classifiers will perform class predictions through their *predict()* methods. The predictions generated by the base classifiers will be submitted to the final classifier, which is responsible for generating the final classification. This classifier is defined through the $final\_estimator$ parameter. The prediction generated by the $final\_estimator$ classifier is considered the final classification of the stacking method.

```
from sklearn.ensemble import StackingClassifier

method = StackingClassifier(estimators=[('name1', classifier1),('name2', classifier2)],
                            final_estimator=classifier, stack_method='predict')

method.fit(training_data_samples, training_data_labels)

predicoes = method.predict(testing_data_samples)
```

Figure 1.12. Ensemble stacking application example in Python using the scikit-learn library.

1.3.4. Practical simulation experiment with machine learning techniques for intrusion detection

Next, details are presented regarding the proposal for a practical simulation experiment with the IoTID20 dataset to evaluate the techniques presented in the section in an intrusion detection scenario with IoT traffic. First, a brief discussion of the existing datasets for intrusion detection simulation is presented.

1.3.4.1. Datasets

This section presents some datasets commonly used in research related to intrusion detection in IoT environments. The objective is to provide a survey of the validation strategies used in state of the art and bring an updated list that can serve as a basis for future researchers, providing indications that can help in deciding which datasets are most suitable for the context of their respective areas. In addition, the dataset chosen to be used in this mini-course is highlighted.

Table 1.1 presents the main databases used in intrusion detection problems. In addition, the main characteristics of the datasets are presented below. The **N.F.** column indicates the **N**umber of **F**eatures and the **L.** column indicates whether the dataset has labels for all records.

Table 1.1. Datasets for evaluating intrusion detection methods.

Dataset	Year	N.F.	L.	IoT	Comments
NSL-KDD [Tavallaee et al. 2009]	2009	42	✓	no	Reduces KDD Cup 99 redundancy problems, but is too old to represent current network standards environment
CTU-13 Botnet [García et al. 2014]	2013	33	✓	no	Focused only on Botnet attacks
RPL-NIDS17 [Verma and Ranga 2019]	2017	21	✓	✓	Differential focus on routing attacks, however, does not address other common IoT attacks
CICIDS-2017 [Sharafaldin et al. 2018]	2017	80	✓	no	It does not include some specific IoT features
CICIDS-2018 [Sharafaldin et al. 2018]	2018	80	✓	no	It does not include some specific IoT features
N-BaIoT [Meidan et al. 2018]	2018	115	no	✓	Only botnet attacks
DS2OS [Aubet 2018]	2018	13	✓	✓	Recent dataset focused on IoT
ToN-IoT [Alsaedi et al. 2020]	2019	7	✓	✓	Recent dataset focused on IoT
Bot-IoT [Koroniotis et al. 2019]	2018	46	✓	✓	Recent data set focused on IoT, but lacks some types of attacks
IoT-23 [Garcia et al. 2020]	2020	21	✓	✓	Real IoT environment traffic, but lacks some types of attacks
IoTID20 [Ullah and Mahmoud 2020a]	2020	12	✓	✓	Recent dataset focused on IoT
MQTT-IoT-IDS2020 [Hindy et al. 2021]	2020	44	✓	✓	Recent data set focusing on IoT, however, exclusively on the MQTT protocol
MQTTset [Vaccari et al. 2020]	2020	33	✓	✓	Recent data set focusing on IoT, however, exclusively on the MQTT protocol
NetFlow Datasets [Sarhan et al. 2021]	2021	43	✓	✓	Recent dataset, has some subsets in the context of IoT, feature standardization

The database used in this minicourse was IoTID20 [Ullah and Mahmoud 2020a]. This choice is mainly because the base has IoT traffic, is fully labeled, and is reasonably small compared to other datasets, which usually have millions of records. This will facilitate the execution of the experiments proposed in the minicourse.

It is one of the latest intrusion detection bases focused on IoT devices. Which was generated through a combination of IoT devices and interconnection structures, simulating a typical smart home environment having two IoT devices, namely, a smart speaker SKT NGU and an EZVIZ Wi-Fi camera [Ullah and Mahmoud 2020b].

These two IoT devices were connected to a home Wi-Fi router, which in turn interconnected with other devices connected to Smart Home, such as laptops, tablets, and smartphones. IoT SKT NGU and EZVIZ devices are victim devices and all other malicious devices. Figure 1.13 shows a simplified version of the architecture of the test environment.

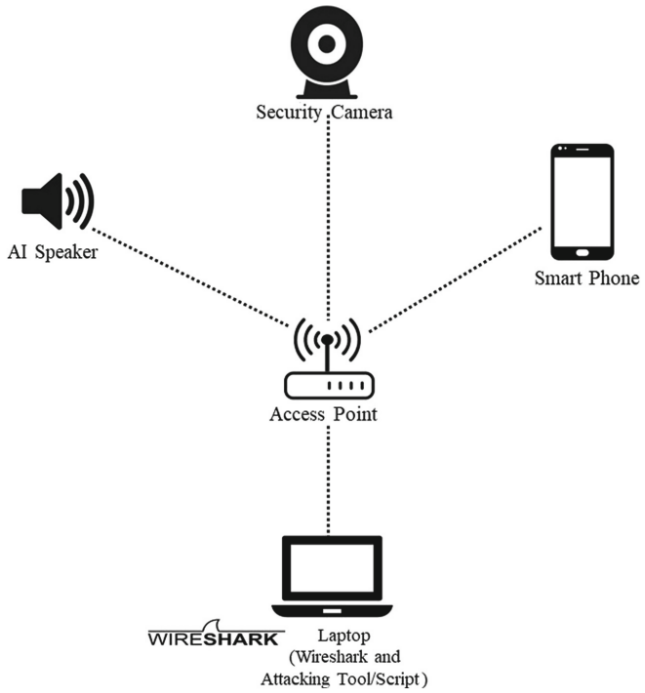

Figure 1.13. The Figure shows the architecture of the monitored environment for creating the IoTID20 dataset [Ullah and Mahmoud 2020b].

The IoTID20 dataset has 80 attributes of network characteristics and three attributes corresponding to labels. The first label is binary, the second corresponds to attack categories, and the last to attack subcategories. In Table 1.2, it is possible to observe the number of instances of traffic present in each type of label.

Table 1.2. Information about the number of instances of the IoTID20 dataset. Information is presented separated by normal traffic, anomalous traffic, their classes and subclasses.

Binary	Instances	Category	Instances	Subcategory	Instances
Benign	40073	Benign	40073	Benign	40073
Anomalous	585710	DoS	59391	DoS	59391
		Mirai	415677	*Ack Flooding*	55124
				Brute force	121181
				HTTP Flooding	55818
				UDP Flooding	183554
		MITM	35377	MITM	35377
		Scan	75265	*Host Port*	22192
				Port OS	53073

Figure 1.14 presents the implementation for reading the dataset for a DataFrame Pandas structure. DataFrame is a 2-dimensional labeled data structure with columns of potentially different types.

```
url_dataset = 'IoTID20/IoT Network Intrusion Dataset.csv'

dataset = pd.read_csv(url_dataset, header=0)
```

Figure 1.14. Example of reading the dataset into a Pandas DataFrame structure.

As part of the data pre-processing, the flow identifiers such as IDs, source IP, destination IP, and timestamps are dropped to avoid learning bias towards attacking and victim-end nodes. Also, the "Cat" label will be considered in this work. The traffic will be classified as benign or in some attack category: DoS, Mirai, MITM, or Scan. The other two labels ("Label" and "Sub_cat") will be taken from the dataset. Figure 1.15 presents an example of removing these columns from the DataFrame.

```
dataset.pop('Timestamp')
dataset.pop('Flow_ID')
dataset.pop('Src_IP')
dataset.pop('Dst_IP')
dataset.pop('Label')
dataset.pop('Sub_Cat')
```

Figure 1.15. Example of removing columns from the DataFrame.

Figure 1.16 presents the implementation for transforming categorical columns into numerical ones. In the case of the IoTID20 dataset, it will only be necessary to transform the labels column (Cat).

```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(dataset['Cat'])
dataset['Cat']= le.transform(dataset['Cat'])
```

Figure 1.16. Example of reading the dataset into a Pandas DataFrame structure.

Next, sanitizing the dataset and removing records with infinite values (inf) and invalid or missing values (NaN) is necessary. For this, the $pd.option\_context$ command will be used to temporarily define options in the context within a block of code. And the option $'mode.use\_inf\_as\_na'$ is used to consider all infinite values (inf) as (NaN). Within this block, the $df.dropna(inplace = True)$ method is used to remove records with NaN from the dataset, as seen in Figure 1.17.

```
with pd.option_context('mode.use_inf_as_na', True):
  dataset = dataset.dropna()
```

Figure 1.17. Example in Python for removing records with infinite and invalid or missing values.

Another important task in dataset pre-processing is standardization. It helps to improve the performance of some classifiers based on machine learning that needs their resources to be normally distributed. They can misbehave if the individual features don't more or less resemble standard normally distributed data. The method used to standardize the data was standard scaling, given by the equation 4, where x is the sample, u is the mean, and s is the standard deviation. The mean and standard deviation are obtained based on the statistic for each attribute in the data set.

$$z = \frac{x - u}{s} \tag{4}$$

Figure 1.18 presents a basic implementation to standardize the data set using the Scikit-learn library. The *fit()* method computes the mean and standard deviation to be used for later scaling and the *transform()* performs standardization by centering and scaling.

```
from sklearn.preprocessing import StandardScaler

labels = dataset.pop('Cat')

scaler = StandardScaler()
scaler.fit(dataset.values)
dataset = scaler.transform(dataset.values)

dataset =  pd.DataFrame(dataset)
dataset['Cat'] = labels
```

Figure 1.18. Implementation example for dataset standardization.

Finally, it is necessary to separate the column of labels (*Cat*) from the dataset into a separate structure called *data_y*. The other columns correspond to traffic attributes and are assigned to *data_x*. This can be done according to Figure 1.19.

```
data_y = dataset.pop('Cat').values
data_x = dataset.values
```

Figure 1.19. Implementation example for label separation.

Once the preparation of the dataset for the experiment is complete, some information on the metrics commonly used to evaluate the ML method in intrusion detection experiments is presented below.

1.3.4.2. Metrics

The evaluation of detection methods is essential to examine the feasibility of applying them in a real environment. One of the evaluations that can be performed is the extraction of detection metrics through experiments, where the methods are trained and tested. The classifications of database events performed by the methods in these experiments can be categorized in the terms presented below.

- **False Negative (FN):** Events classified as normal by the detection method and which are intrusions;
- **False Positive (FP):** Non-intrusive events classified as intrusive by the intrusion detection technique;
- **True Negative (TN):** This category covers non-intrusive events, which were correctly classified by the detection method;
- **True Positive (VP):** This class includes events correctly reported as intruders by the intrusion detection technique.

From these terms, it is possible to construct confusion matrices. They consist of tables that present a summary of the classification performed by the method, indicating the number of events classified in relation to the predicted class and the true class of the element. The confusion matrix itself is not a metric for evaluating classification methods. However, several analyzes and metrics can be made from such information. In Table 1.3, it is possible to observe an example of a confusion matrix.

Original Label	**Prediction**	
	Normal (+)	Ataque (-)
Normal (+)	VN	FP
Ataque (-)	FN	VP

Table 1.3. Example of a confusion matrix.

Based on the abovementioned terms, it is possible to calculate different metrics that help evaluate detection methods built by machine learning algorithms. Below are the main metrics used [Liu and Lang 2019].

1. **Accuracy:** This metric corresponds to the proportion of correctly classified instances in relation to the total number of existing instances. It may not be a good aspect to consider in cases of large class imbalance. The accuracy is calculated from the Equation 5, presented below:

$$ACC = \frac{VP + VN}{VP + VN + FP + FN} \tag{5}$$

2. **Precision:** Another widely used metric for evaluating machine learning methods. This rate indicates the proportion of instances correctly detected as intrusive out of all those detected as intrusive, as can be seen in Equation 6.

$$PRE = \frac{VP}{VP + FP} \tag{6}$$

3. **Recall:** Also known as sensitivity, consists of the number of instances correctly classified as intrusive among all intrusive instances, calculated according to Equation 7.

$$Recall = \frac{VP}{VP + FN} \tag{7}$$

4. **F1-Score:** The F1 score is the harmonic mean of precision and recall, where an F1 score is best at 1 (perfect precision and recall) and worst at 0. The formula for the F1 score is given in Equation 8.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{8}$$

5. **Balanced Accuracy (BACC):** Balanced Accuracy is an interesting metric to evaluate detection performance on unbalanced datasets. It is defined as the average of the recall obtained in each class, as seen in Equation 9. Where R_i corresponds to the recall (R) obtained considering the i-th (i) class present in the dataset, and n is the number of existing classes.

$$Balanced\ Accuracy = \frac{\sum_{i=1}^{n} R_i}{n} \tag{9}$$

Figure 1.20 shows the basic Python implementation for calculating detection metrics using the methods provided by the Scikit-learn library.

```
import sklearn.metrics

acc = sklearn.metrics.accuracy_score(testing_data_labels, predictions)
print('Accuracy: {}'.format(acc))

acc_balanced = sklearn.metrics.balanced_accuracy_score(testing_data_labels, predictions)
print('Accuracy balanced: {}'.format(acc_balanced))

precision = sklearn.metrics.precision_score(testing_data_labels, predictions, average=None)
print('Precision: {}'.format(precision))

reacll = sklearn.metrics.recall_score(testing_data_labels, predictions, average=None)
print('Recall: {}'.format(recall))

f1_score = sklearn.metrics.f1_score(testing_data_labels, predictions, average=None)
print('F1_score: {}'.format(f1_score))
```

Figure 1.20. Example of implementation of detection metrics using the Scikit-learn library.

1.3.4.3. Experiments

Intrusion detection databases are large datasets that can be used to validate and evaluate detection models. These data can be used to train and test the models. However, ideally, models should be evaluated with samples not used to build, train, or tune the model. In order to obtain an unbiased assessment of the model's effectiveness.

Thus, it is proposed in this work to carry out experiments with the Hold-Out 70-30 technique to divide the data set and carry out the evaluation. In this strategy, the data set (*data_x* and *data_y*) is divided into 70% of the data for training (*training_x* and *training_y*) the models and 30% of the data for testing ($test_x$ and $test_y$) the performance of the generated models. This division is performed to prevent data used to train the model from being used to test it. Furthermore, a stratified strategy divides these data so that the training and test sets maintain equal proportions of instances per class.

Figure 1.21 presents the basic implementation in Python for experimenting with the Hold-Out 70-30 technique and the pre-processed IoTID20 dataset. In this experiment, the data is split, and the generated training data is used to train the detection method.Before the training, it is necessary to define which classification method will be used in the experiment. For this, it is necessary to assign the classifier to the *method* variable. This can be done according to the definitions presented in each of the previous sections. After training, the test data, without the labels, is submitted to the method for classification. From this, the generated classifications can be used to calculate the detection metrics of the method. The example shows the calculation of the accuracy metric, however the other metrics presented can also be used in the experiment.

```
from sklearn.model_selection import train_test_split

training_X,test_x,training_y,test_y = train_test_split(data_X,data_y,test_size=0.3)

#Definition of the classification method to be used

method.fit(training_X, training_y)

predictions = method.predict(test_x)

acc_balanced = sklearn.metrics.balanced_accuracy_score(test_y, predictions)
print('Accuracy balanced: {}'.format(acc_balanced))

#Other metrics can be used below
```

Figure 1.21. Example of experiment implementation using the hold-out 70-30 technique.

The objective is to use the points mentioned in this practical section to implement, execute, evaluate, and compare each of the presented machine learning methods.

1.4. State of the Art in intrusion detection in the context of IoT/Fog/Cloud

In state-of-the-art, several works proposed intrusion detection approaches for IoT, Fog, and Cloud environments. Some approaches are based on analysis by signature and others by behavior. In addition, some works have inserted the detection module in the layer of IoT devices and others in the upper layers. Some of these approaches are presented and discussed below.

In signature-based detection, monitored actions are compared to predefined intrusive events. Signature-based solutions enable rapid detection and reduce the occurrence of false alarms. Thus, they are interesting options to be deployed on [Arshad et al. 2019] IoT devices. However, they have the limitation of not being able to detect attacks that do not have a signature known to the IDS.

Most state-of-the-art works have proposed behavior-based detection approaches. Some works consider a branch of analysis by a behavior called analysis by the specification. This type of solution employs rules and thresholds that define the expected default behavior for monitored components. It is similar to anomaly detection, and both detect intrusions when network behavior deviates from specified. The main difference is that in specification-based analysis, a human expert sets the rules [Mitchell and Chen 2014]. [Yaseen et al. 2017] proposed a threshold-based approach for detecting selective forwarding attacks on Wireless Sensor Networks (WSNs). In [Aliyu et al. 2018], a challenge or response based fog detection approach is proposed. The detection nodes periodically interrogate nearby nodes, sending interrogation packets and waiting for a response according to a previously specified calculation. The major disadvantages of this type of analysis are the necessary specificity and the need for a human expert to define the system's expected behavior.

Anomaly detection approaches, on the other hand, are usually modeled using Machine Learning (ML) techniques. There are threats against IoT devices and even against services deployed in the fog and cloud layer that need robust methods to be detected. These complex detection approaches often cannot be applied to IoT devices due to their computational constraints [Arshad et al. 2019]. In addition, approaches that only perform analysis on IoT nodes only consider a restricted view of events, thus limiting their ability to deal with complex, multi-stage, and distributed attacks. In this way, research on detection techniques implemented in devices in the fog layer has been growing, seeking to avoid the problem of latency, implement a distributed strategy, and take advantage of the privileged position of these devices. Many works have proposed solutions deployed in the fog to perform network monitoring of IoT devices through network packet analysis. This location allows the IDS to have a global view of the IoT network, monitoring all traffic [Souza et al. 2020, Lawal et al. 2021, Labiod et al. 2022].

The previously presented KNN method has been used in some works due to its good classification performance. In [Lawal et al. 2021], the proposed mitigation framework for fog computing uses a database that stores signatures of previously detected attacks and an anomaly-based detection scheme that uses the K-Nearest Neighbor (KNN) classification algorithm to detect DDoS attacks. The Euclidean, Manhattan, and Chebychev distances were evaluated, and the first two achieved the best results. An existing disadvantage in KNN is the computational cost, which can become high because it is

necessary to compare the new instances with those stored in the example base. The authors [Souza et al. 2020] proposed a hybrid approach called DNNKNN, composed of a DNN model and the K-Nearest Neighbor algorithm, to improve the method's robustness and reduce the prediction cost of the kNN algorithm. This approach made it possible to maintain a good detection performance and obtain a reduction of approximately 90% of the KNN prediction cost.

Another technique commonly used in intrusion detection research is the Deep Neural Network (DNN). In state-of-the-art, it is possible to find several works that propose detection approaches based on DNN. Most architectures consider two hidden layers. Inserting more layers did not bring detection benefits [Lalouani and Younis 2021]. In addition, architectures with a greater number of layers are more complex and demand more costs for the training process. [Sahar et al. 2021] used hidden ReLu layers with 768 and 512 neurons, respectively. However, a large number of neurons per layer may also be unnecessary. Several other works sought to build less complex neural models with fewer neurons. [Kumar et al. 2021b] used two hidden layers with 32 and 16 neurons, respectively, with an activation function ReLU. In [Lalouani and Younis 2021], the function used in the two hidden layers was the ReLU in the 64 neurons in each layer. In [Kumar and Tripathi 2021], a layer with 16 neurons and another with 12 neurons were used, with ReLU activation function. ReLU is more efficient for training large-scale data in terms of time and cost [Sahar et al. 2021]. On the other hand, the Soft-Max activation function is one of the most used in the output layer [Souza et al. 2020, Lalouani and Younis 2021, Kumar and Tripathi 2021, Kumar et al. 2021b]. Several other techniques based on neural networks can be used for intrusion detection and have state-of-the-art works. However, this work focuses only on feed-forward models.

One of the major challenges regarding neural model-based detection approaches is the high cost of training complex deep learning architectures during model updates. Motivated by this challenge, some works used the distributed characteristic of fog computing to propose distributed training approaches between fog devices through information sharing [Diro and Chilamkurti 2018, Labiod et al. 2022].

The authors [Diro and Chilamkurti 2018] proposed a new distributed approach based on DNN feed-forward to detect intrusions in the IoT environment. The approach is deployed distributed in fog devices and has two levels. It uses a fog device as the master responsible for training and placing the model on the other fog nodes. Second-level nodes send model updates to the master node. The master node updates the global model and spreads the updates to other nodes. The experiment demonstrated that the distributed approach could detect cyber attacks better than centralized algorithms due to the sharing of parameters that can avoid local minima in training [Diro and Chilamkurti 2018]. However, this primary node can be considered a single point of failure (SPOF), which is easier to compromise than a cloud-based parameter update approach.

In this context of distributed training of detection models, it is important to discuss Federated Learning (FL), which consists of a strategy to form a global ML model from several local data-driven models [Lalouani and Younis 2021]. This strategy decentralizes machine learning, eliminating the need to gather data on a single device. Instead, the model is trained through multiple iterations on different devices. FL is an ideal framework

for aggregating distributed models, preserving privacy and allowing convergence to a distributed learning engine with precision close to that of a centralized implementation. The FLIDS [Lalouani and Younis 2021] approach employs federated learning to enable privacy-preserving distributed aggregation in training deep neural models deployed in fog computing.

The framework architecture proposed by [Rey et al. 2022] is composed of clients that monitor IoT devices and a server that coordinates a Federated Learning process. The approach provides for intrusion detection through neural network models. Considering that IoT devices generally have limited resources and modest reliability, the clients responsible for training the models are not the devices to be protected, but other entities capable of collecting traffic from IoT devices present on the same network, such as fog nodes [Rey et al. 2022].

[Abbasi et al. 2021] presents extensive research on DL methods to detect anomalies in network traffic. They point out that federated learning approaches are promising to overcome the challenges of training a DL approach with many samples and training parameters in environments with resource constraints, such as IoT and fog environments. However, they are also susceptible to threats, model poisoning attacks can be carried out through updates of corrupted models sent to the server. Furthermore, due to the privacy issues employed in FL, it is difficult to verify whether the received models really correspond to the local training data or not [Lalouani and Younis 2021]. Furthermore, using a device as a server centralizes the process of aggregating models on a device. This centralization can bring some concerns, such as the need to trust a central device and the possibility of this central device becoming a single point of failure, where the failure or compromise of this device by an attacker, could harm the collaboration network completely.

In addition to deep neural methods, research on ensemble approaches is also promising for intrusion detection. Classifiers based on Ensemble Learning can be proposed to improve adaptability and generalizability in multiclass classification. Several recent works have investigated the use of the ensemble Random Forest method in intrusion detection approaches [Illy et al. 2019, Farukee et al. 2020, Kumar et al. 2022]. The authors [Farukee et al. 2020], however, used RF to select the main characteristics of the traffic to be submitted to another classifier. An important aspect of RF is the calculation of the importance of the resource. The Gini impurity criterion index is used. Thus, they used the RF property to classify resources according to their importance. Prioritizing accuracy, features with feature importance less than 0.005 were discarded. The RF used in the ensemble method proposed by Kumar et al. [Kumar et al. 2021a] had 100 estimators, maximum depth equal to 3, a minimum number of examples for split equal to 10, a minimum number of samples needed to be a leaf node equal to 6 and Entropy criteria. [Hosseini and Sardo 2022] proposed an approach with RF classification and feature selection by Spider-Monkey Optimization (SMO). The authors [Kumar et al. 2020b] evaluated RF in the context of DDOS attack detection and found that the fact that RF is insensitive to outliers, missing values, overfitting, and having the ability to handle a large number of incoming traffic makes it suitable in the process anomaly detection tool for the blockchain-IoT environment. However, no details are provided regarding the number of trees used in the structure. The performance of the proposed distributed structure is evaluated using a BoT-IoT dataset. The proposed distributed structure with RF and 10

base attributes surpasses some current state-of-the-art techniques, reaching recall close to 100% in all classes. However, it is observed that several techniques have already presented similar performance with this Bot-IoT dataset.

The Extra Tree, a method very similar to RF, was also used in detection approaches. In ET, randomness goes a step further in how divisions are calculated. Instead of looking for the most discriminating cut points, they are drawn randomly in ET, making the ET training process faster. In [Albdour et al. 2020] is proposed an ET-based intrusion detection approach for the fog layer. The approach has 10 DTs and uses the Gini criterion. The approach showed 98.3% accuracy with the UNSW-NB15 dataset. However, considering only the binary detection, not being able to identify the attack categories. In [Souza et al. 2022a] ET was used in a first level of binary detection, obtaining a high detection rate. The ET used is composed of 10 estimators DT, the minimum sample size for division is 2, the number of attributes considered for better division is the root of the number of existing attributes, and the Gini Index is used as a criterion.

Following the line of ensemble classifiers with Decision Tree, [Lawal et al. 2020] presented an approach with Extreme Gradient Boosting (XGBoost) for anomaly detection in an IoT framework. XGBoost is based on the ensemble boosting technique, where weak classifier predictions are combined to develop a strong classifier, employing additive techniques. In addition to the speed and performance benefits of XGBoost, additional advantages include the avoidance of overfitting and the full utilization of [Lawal et al. 2020] computational resources. In the experiments performed by [Lawal et al. 2020] with the Bot-IoT dataset, XGBoost was able to achieve 99.96% average accuracy, 98% average recovery, 97% average accuracy, and 97% average f1-score in the multiclass ranking. Superior performance compared to other classifier algorithms such as DT, kNN, and Naive Bayes.

Several proposed papers have also focused on ensemble voting approaches. The authors [Illy et al. 2019] proposed a voting-based ensemble approach composed of KNN, RF, DT bagging, and DT boosting. [Alhowaide et al. 2021] report two main strategies to combine the results of the base classifiers in ensemble methods by voting: hard and soft voting. They highlight that soft voting can perform better than hard voting because it takes into account more information and uses the uncertainty of each classifier in the final decision.

The ensemble technique proposed in [Al-Khafajiy et al. 2021] is composed of three base classifiers which together provide analysis of collected traffic for intrusion detection. The decisions generated by the three classifiers are combined using the majority voting rule (hard), where the class with the highest number of votes of the three classification systems is defined as the final classification. The authors point out that the majority voting rule is the simplest and most effective voting scheme in this case.

In [Souza et al. 2022a], a binary detection is applied, and only the events detected as intrusive are submitted for multiclass analysis. The proposed multiclass analysis model is a soft voting set comprising three classification models: ET, RF, and DNN. The combination strategy employed was soft voting, that is, the complete method implements a combinator that predicts the class label based on the argmax of the sums of the probabilities predicted by each of the three classification models.

In addition, the proposal of methods based on ensemble stacking strategy also stands out in state of the art. [Kumar et al. 2020a] proposed an approach where the KNN, XGBoost, and Naive Bayes classifiers are trained in parallel and act as basic classifiers. As a result, three prediction results $P1$, $P2$, and $P3$ are obtained, which an RF uses for the final classification. Thus, the RF is responsible for learning to combine the individual results of each weak classifier into a final overall result. The experiments with the data set showed that the approach performed well but presented difficulties in identifying some attacks.

The authors Kumar et al. [Kumar et al. 2021a] also present another similar approach; in this case, they proposed an ensemble method with Naive Bayes, DT, RF, and XGBoost to detect attacks on Internet of Medical Things (IoMT) networks. Naive Bayes, DT, and RF classifiers operate in parallel on the first level. As a result, three prediction outputs $P1$, $P2$, and $P3$ are obtained and are used by XGBoost to build the final predictive model.

Ensemble techniques can be useful in intrusion detection, allowing you to build a strong classifier to identify a specific attack's specific class. However, as these techniques use several classifiers, they may present difficulties related to the processing and training time of the models.

Some of the approaches mentioned above result in classifying events as normal or malicious, making it impossible to identify the type of attack. The methods that perform only the identification that an intrusion has occurred, that is binary detection, are not enough to provide efficient security. The mechanism must be able to mitigate the invasion so that it does not succeed. Therefore, it is important to classify the attack in its category so that specific countermeasures are executed for the given type of threat. In addition, the classification of the type or category of the attack is important for the decision-making of the person responsible for the network.

It is essential to identify more information about the attack so that specific countermeasures can be carried out for each type of threat. For example, a probing attack is usually performed before more powerful attacks such as DoS, DDoS, remote access attacks, etc [Nguyen et al. 2019]. Thus, running additional detection mechanisms to reinforce security when detecting a probing category threat may be interesting. Also, classifying attack types or category is important for the network manager. From identifying the category of a certain attack that occurs with a specific frequency, the person responsible for the network can decide to implement actions to correct the vulnerability used by the attack.

To improve the accuracy of multiclass detection without overloading the IoT-Fog environment, the approach proposed in [Souza et al. 2020, Souza et al. 2022a] presents a two-step hierarchical detection method. A binary detection analysis (Step 1 - Detection) is performed on fog computing devices to detect intrusive events. Only events detected as intrusive by the first step are sent to the multiclass analysis (Step 2 - Identification) in the cloud. The analysis module of Step 2 is responsible for identifying the attack category and providing further information to the countermeasures module. If the event is detected as non-intrusive by Step 1, it is automatically sent to the module output to free up the flow. The analysis performed at the cloud computing layer aims to classify the event

into a specific attack category or normal behavior. This step allows you to correct first-level false positives. The classifier consists of a more robust method that requires more processing than the first stage's analysis. This method will be activated only when the first level detects the event as intrusive. In this way, it is possible to apply a complex analysis that can more accurately classify the event into a specific class of attack for the execution of countermeasures.

For the binary detection module of the first stage, the authors proposed a hybrid DNNKNN approach previously mentioned [Souza et al. 2020]. A soft voting ensemble analysis was proposed for the second stage, consisting of three robust classification models: ExtraTree, Random Forest, and Deep Neural Network [Souza et al. 2022a]. The architecture also applies attribute selection and class balancing techniques. The obtained results provided superior detection performance than several state-of-the-art approaches.

This hierarchical detection architecture proposed in [Souza et al. 2020] has already served as a basis for other works. In [Labiod et al. 2022], some modifications are proposed in the approaches throughout the architecture, mainly concerning the first level of detection. The VAE-MLP method, a binary detection approach based on Variational AutoEncoder and DNN, was proposed.

As presented, several state-of-the-art works were found. However, there are still many challenges in this context. Next section discusses some important aspects observed in state-of-the-art related to intrusion detection in an IoT/Fog/Cloud context. The objective is to instigate an initial reflection on this research topic's problems, challenges, and open questions.

1.5. Discussions, Reflections and Questions

This section discusses some important aspects observed in state-of-the-art related to intrusion detection in an IoT/Fog/Cloud context. The aim is to instigate an initial reflection on this research topic's problems, challenges, and open questions.

1.5.1. Deployment strategy

The deployment location of the detection solution is an important aspect that must be considered when designing the network or host-based approach. Resource constraints, usually existing in devices inserted in the context of IoT applications, make it difficult to implement robust detection approaches in the devices themselves. However, some works have proposed approaches partially implemented in IoT devices through lighter signatures-based techniques. Other approaches proposed their solutions to operate entirely in the fog computing layer. In addition, some works have also delegated part of the analysis to the cloud computing layer.

The processing time and cost of robust machine learning models can be high, especially when considering strategies that use multiple detection methods. Implementing a robust approach in IoT devices and even in the fog computing layer can be a problem. Robust and slow multiclass analysis performed in the fog can overwhelm the device and slow network flow. It is necessary to investigate new intrusion detection approaches capable of managing and optimizing the analysis process in the various layers of the IoT environment. In other words, optimizing available resources on IoT devices, fog, and cloud.

1.5.2. Detection method category

The categories of detection methods can be anomaly, signature, and specification. Some detection approaches found in the state of the art focus on signature-based detection [Arshad et al. 2019, Lawal et al. 2021]. They are not able to detect new attacks or variations of known attacks.

On the other hand, some works have proposed approaches with analysis by specification, which detects intrusions when the network behavior deviates from the specified [Yaseen et al. 2017, Aliyu et al. 2018]. In this type of detection, a human expert specifies normal behavior. Thus, the approaches based on specifications found are closely linked to specific protocols or attacks. The weakness of these approaches is the difficulty of generalizing the approaches to a broader context with other attacks and protocols.

Finally, anomaly detection considers that all abnormal behavior is an intrusion and can detect new attacks. Several approaches have proposed anomaly-based methods to detect intrusions into the fog computing layer. Usually, they are machine learning-based approaches. IoT devices often have limited computational resources [Ni et al. 2018]. These restrictions make it difficult to conduct analyses based on complex anomaly techniques on IoT devices, thus preventing new attacks [Zarpelão et al. 2017]. Furthermore, anomaly-based approaches may suffer from problems related to false positives. Also, there is only a narrow view of events. Thus, research proposing hybrid detection approaches, combining detection categories, are interesting and necessary to maximize the advantages and minimize the disadvantages of these types of analysis and obtain a complete solution.

1.5.3. Machine Learning approachs

It is essential to categorize the attack to take specific countermeasures for the threat. For example, a probe category attack is usually performed before more powerful attacks such as DDoS, remote access attacks, etc. [Nguyen et al. 2019]. Thus, running additional detection mechanisms to strengthen security when detecting an attack from the probe attack category is interesting. Also, the type of attack or category classification is important to the network owner. From identifying the category of a certain attack that occurs with a specific frequency, the person responsible for the network can decide to implement actions to correct the attacker's vulnerability.

Many approaches found in the state-of-the-art focus on performing binary detection (attack or non-attack). However, binary methods cannot identify the type or category of attack [Albdour et al. 2020]. The approaches, which aim to classify the attack into specific categories, are multiclass. However, it is observed in the state-of-the-art that these approaches have lower accuracy rates than the binary detection methods [Nguyen et al. 2019, Kumar et al. 2020b]. In addition, these approaches may present difficulties related to false-positive problems and low detection of some types of attacks [Diro and Chilamkurti 2018, Kumar et al. 2020b, Kumar and Tripathi 2021].

Several works sought to propose ML-based solutions with more restricted models to respect the resource capacity of the devices involved in this context. The approaches obtained interesting results related to detection; however, single classifiers are subject to inconsistencies and need to be improved in detecting some types of attacks [Diro and Chilamkurti 2018, Kumar and Tripathi 2021].

Individual classifiers may experience instability. There is no guarantee that a classifier will always perform at its best in all situations. However, with Ensemble Learning (EL), a better classification performance than any individual classifier can be achieved [Traganitis et al. 2018]. Classifiers based on DL and ensemble approaches have been the object of research recently and have achieved promising results in intrusion detection. Ensemble methods can be proposed to improve adaptability and generalizability in multiclass classification [Traganitis et al. 2018]. Thus, combining different machine learning models for optimal performance and attack detection is another research trend. However, ensemble methods have greater computational complexity and require more training time and resources. Thus, it is important to consider the characteristics of the devices where the approach will be implemented in the design of detection methods.

In addition, using more complex techniques such as DL and Ensemble also makes efforts to improve training strategies, optimize resources and reduce cost and computational time. Collaborative IDS based on federated learning in fog computing emerges as an interesting alternative to optimize the training of complex DL and ensemble learning approaches.

Another interesting point is that the performance of the methods is related to the quality and quantity of the training data. This can be challenging, as obtaining training data can be extremely arduous. In this context, hybrid approaches that combine supervised machine learning techniques with other techniques that work with unlabeled data, such as unsupervised or reinforcement learning, are promising.

1.5.4. Collaborative IDSs approachs

Collaborative detection approaches are very promising solutions in the IoT-Fog-Cloud context, mainly due to the distributed nature of fog computing. However, these solutions have limitations, such as vulnerability to insider attacks [Li et al. 2021]. Insider attacks occur when attackers compromise a device that is part of the Collaborative Intrusion Detection System (CIDS) and, from there, perform false collaborative actions to undermine the functioning of the collaborative detection approach. As a solution, one can investigate trust management approaches to defend against insider attacks.

Another highlight is that although a server is responsible for aggregating the models, it would be possible to pass the steps from the server to the devices themselves, decentralizing the server into several entities [Rey et al. 2022]. This decentralization would avoid the need to have a trusted device as a central server and hence the single point of failure problem. However, a reliable decentralized approach would require a Blockchain-based architecture to be used as a decentralized database, where each device would share its local model and reliably retrieve models from other devices.

1.5.5. Detection models update

Another research point is training detection models based on ML e DL. This intrusion detection subtopic still has several points that need to be further studied and improved. Machine learning-based detection approaches must be retrained over time to prevent them from becoming obsolete. In the IoT context, the network changes over time. New devices can be inserted, and others removed. Therefore, detection models based on machine learning techniques need to be updated. Considering the new components, new data must be collected from the network to generate an updated model capable of identifying truly abnormal behaviors in the new IoT network. Furthermore, IoT applications can be inserted in a context of high device mobility, which undoubtedly makes the intrusion detection process even more challenging. Thus, another open question is understanding the maximum time a model can operate without becoming obsolete in this IoT context [Abbasi et al. 2021]. The ideal strategy would be to retrain the discovery models whenever there is any change in the network, such as the insertion and removal of devices. However, the training process can cost a lot of resources and overload the network. Machine learning models often suffer from high complexity in the training phase as they consume many resources and time. Devices in the IoT context have resource constraints, so the complexity of the detection models to be retrained can be considered a major challenge. Some works proposed a distributed training mechanism between fog devices with the exchange of parameters of the machine learning method [Diro and Chilamkurti 2018].

Approaches based on Federated Learning are also very interesting [Rey et al. 2022, Lalouani and Younis 2021]. However, they are also susceptible to threats. Model poisoning attacks can be carried out through corrupted model updates sent to the server. Trust management approaches can be investigated to defend against these insider attacks. Furthermore, due to the privacy concerns employed in FL, it isn't easy to verify whether the received models match the local training data or not [Lalouani and Younis 2021]. However, more studies are needed to propose detection approaches that are less susceptible to changes in the network and lighter to optimize detection models in terms of time com-

plexity and resource consumption so as not to overload fog computing.

Another important point is that machine learning approaches' good detection performance depends on the training data's quality and quantity. The vast majority of works found proposed approaches based on supervised learning. To train these approaches, having a large number of labeled data is usually necessary. This becomes an issue as you need to capture network traffic from the network and label it for use in the detection model training process. Therefore, this task can become extremely costly due to the large amount of data required. Some works have proposed promising approaches for the training data labeling problem, using data sampling and clustering methods [Ravi and Shalinie 2020]. As points for future studies, the need for research on new hybrid approaches based on supervised and unsupervised learning to overcome the problem of data labeling stands out. Therefore, proposing approaches that combine supervised machine learning techniques with other techniques capable of working with unlabeled data, such as unsupervised learning or reinforcement learning, can be a great solution.

Updating detection approaches is a major challenge, which generates the need for further studies to find the ideal strategy. They must consider how often it takes to update the model, training costs, data acquisition, and labeling.

1.6. Conclusion

IoT is spreading in all areas due to its ability to make objects smart. In this way, they can monitor and act in the environment in which they operate. IoT devices have limited resources and must send information to places with more computing resources. Fog computing then emerged as an excellent processing solution close to devices. IoT and fog are not free from security threats and vulnerabilities. Added to the significant damage generated by attacks in this environment, this fact generates the need to concentrate efforts in this area. Intrusion detection systems are an essential tool to ensure IoT security.

In this mini-course, the fundamental concepts involved in the theme of this work were presented, the main threats present in IoT environments were discussed, and concepts related to Intrusion Detection Systems were introduced. In addition, several machine learning techniques that can be used to analyze and detect intrusions were presented. This section also proposes to conduct simulation experiments with the IoTID20 dataset to evaluate the machine learning techniques presented in an intrusion detection scenario with IoT traffic. In Section 1.4, state-of-the-art surveying through a literature review is exposed, and the approaches proposed by the main related works are presented. Finally, Section 1.5 discusses some important aspects observed in state-of-the-art related to intrusion detection in an IoT/Fog/Cloud context. The objective is to instigate an initial reflection on this research topic's problems, challenges, and open questions.

For future research, the following points are highlighted: (1) investigate hybrid approaches combining detection categories; (2) investigate solutions capable of managing the analysis process in the various layers of the IoT-Fog-Cloud environment; (3) investigate new approaches for multiclass detection to achieve accuracy greater than or similar to binary detection; (4) investigate ensemble and hybrid methods to improve multiclass detection considering resource constraints; (5) investigate strategies to update/retrain detection models with federated learning.

References

[Abbasi et al. 2021] Abbasi, M., Shahraki, A., and Taherkordi, A. (2021). Deep learning for network traffic monitoring and analysis (ntma): A survey. *Computer Communications*, 170:19–41.

[Ahmad et al. 2021] Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., and Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150.

[Al-Fuqaha et al. 2015] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376.

[Al-Khafajiy et al. 2021] Al-Khafajiy, M., Otoum, S., Baker, T., Asim, M., Maamar, Z., Aloqaily, M., Taylor, M., and Randles, M. (2021). Intelligent control and security of fog resources in healthcare systems via a cognitive fog model. *ACM Trans. Internet Technol.*, 21(3).

[Albdour et al. 2020] Albdour, L., Manaseer, S., and Sharieh, A. (2020). Iot crawler with behavior analyzer at fog layer for detecting malicious nodes. *Int. J. Commun. Networks Inf. Secur.*, 12(1).

[Alhowaide et al. 2021] Alhowaide, A., Alsmadi, I., and Tang, J. (2021). Ensemble detection model for iot ids. *Internet of Things*, 16:100435.

[Aliyu et al. 2018] Aliyu, F., Sheltami, T., and Shakshuki, E. M. (2018). A detection and prevention technique for man in the middle attack in fog computing. *Procedia Computer Science*, 141:24 – 31. The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018) / The 8th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2018) / Affiliated Workshops.

[Alsaedi et al. 2020] Alsaedi, A., Moustafa, N., Tari, Z., Mahmood, A., and Anwar, A. (2020). Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150.

[Anderson 1980] Anderson, J. P. (1980). Computer security threat monitoring and surveillance. *Technical Report, James P. Anderson Company*.

[Arshad et al. 2019] Arshad, J., Azad, M. A., Abdellatif, M. M., Rehman, M. H. U., and Salah, K. (2019). Colide: a collaborative intrusion detection framework for internet of things. *IET Networks*, 8(1):3–14.

[Atzori et al. 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.

[Aubet 2018] Aubet, F.-X. (2018). *Machine Learning-Based Adaptive Anomaly Detection in Smart Spaces*. PhD thesis.

[Aversano et al. 2021] Aversano, L., Bernardi, M. L., Cimitile, M., and Pecori, R. (2021). A systematic review on deep learning approaches for iot security. *Computer Science Review*, 40:100389.

[Bace and Mell 2001] Bace, R. and Mell, P. (2001). Nist special publication on intrusion detection systems. Technical report, BOOZ-ALLEN AND HAMILTON INC MCLEAN VA.

[Berry et al. 2019] Berry, M. W., Mohamed, A., and Yap, B. W. (2019). *Supervised and unsupervised learning for data science*. Springer.

[Birkinshaw et al. 2019] Birkinshaw, C., Rouka, E., and Vassilakis, V. G. (2019). Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks. *Journal of Network and Computer Applications*, 136:71 – 85.

[Bonomi et al. 2012] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM.

[Boukerche et al. 2007] Boukerche, A., Machado, R. B., Jucá, K. R., Sobral, J. B. M., and Notare, M. S. (2007). An agent based and biological inspired real-time intrusion detection and security model for computer network operations. *Computer Communications*, 30(13):2649–2660.

[Breiman 1996] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

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

[Breiman et al. 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

[Camhi 2015] Camhi, J. (2015). Former cisco ceo john chambers predicts 500 billion connected devices by 2025. *Business Insider*.

[Campello and Weber 2001] Campello, R. S. and Weber, R. F. (2001). Sistemas de detecçao de intrusao. *Minicurso procedente do 19º Simpósio Brasileiro de Redes de Computadores*.

[Chua and Yang 1988] Chua, L. O. and Yang, L. (1988). Cellular neural networks: Applications. *IEEE Transactions on circuits and systems*, 35(10):1273–1290.

[Cunningham et al. 2000] Cunningham, P., Carney, J., and Jacob, S. (2000). Stability problems with artificial neural networks and the ensemble solution. *Artificial Intelligence in Medicine*, 20(3):217 – 225.

[Dalton and Deshmane 1991] Dalton, J. and Deshmane, A. (1991). Artificial neural networks. *IEEE Potentials*, 10(2):33–36.

[Diro and Chilamkurti 2018] Diro, A. A. and Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, 82:761 – 768.

[Farukee et al. 2020] Farukee, M. B., Shabit, M. Z., Haque, M. R., and Sattar, A. S. (2020). Ddos attack detection in iot networks using deep learning models combined with random forest as feature selector. In *International Conference on Advances in Cyber Security*, pages 118–134. Springer.

[García et al. 2014] García, S., Grill, M., Stiborek, J., and Zunino, A. (2014). An empirical comparison of botnet detection methods. *Computers & Security*, 45:100 – 123.

[Garcia et al. 2020] Garcia, S., Parmisano, A., and Erquiaga, M. J. (2020). Iot 23: A labeled dataset with malicious and benign iot network traffic.

[Garcia-Morchon et al. 2013] Garcia-Morchon, O., Kumar, S., Keoh, S., Hummen, R., and Struik, R. (2013). Security considerations in the ip-based internet of things draft-garciacore-security-06. *Internet Engineering Task Force*.

[Geurts et al. 2006] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63(1):3–42.

[Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Haykin 2001] Haykin, S. S. (2001). *Redes neurais: Princípios e Práticas*. Bookman.

[Heady et al. 1990] Heady, R., Luger, G., Maccabe, A., and Servilla, M. (August 1990). The architecture of a network level intrusion detection system. Technical report, University of New Mexico, Department of Computer Science.

[Hindy et al. 2021] Hindy, H., Bayne, E., Bures, M., Atkinson, R., Tachtatzis, C., and Bellekens, X. (2021). Machine learning based iot intrusion detection system: An mqtt case study (mqtt-iot-ids2020 dataset). In Ghita, B. and Shiaeles, S., editors, *Selected Papers from the 12th International Networking Conference*, pages 73–84, Cham. Springer International Publishing.

[Hosseini and Sardo 2022] Hosseini, S. and Sardo, S. R. (2022). Network intrusion detection based on deep learning method in internet of thing. *Journal of Reliable Intelligent Environments*, pages 1–13.

[Illy et al. 2019] Illy, P., Kaddoum, G., Moreira, C. M., Kaur, K., and Garg, S. (2019). Securing fog-to-things environment using intrusion detection system based on ensemble learning. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7.

[Iorga et al. 2018] Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N. S., and Mahmoudi, C. (2018). Fog computing conceptual model. Technical report.

[Ke et al. 2017] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA. Curran Associates Inc.

[Kolias et al. 2016] Kolias, C., Stavrou, A., Voas, J., Bojanova, I., and Kuhn, R. (2016). Learning internet-of-things security" hands-on". *IEEE Security & Privacy*, 14(1):37–46.

[Koroniotis et al. 2019] Koroniotis, N., Moustafa, N., Sitnikova, E., and Turnbull, B. (2019). Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796.

[Kumar et al. 2020a] Kumar, P., Gupta, G. P., and Tripathi, R. (2020a). A distributed ensemble design based intrusion detection system using fog computing to protect the internet of things networks. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–18.

[Kumar et al. 2021a] Kumar, P., Gupta, G. P., and Tripathi, R. (2021a). An ensemble learning and fog-cloud architecture-driven cyber-attack detection framework for iomt networks. *Computer Communications*, 166:110–124.

[Kumar et al. 2020b] Kumar, P., Kumar, R., Gupta, G. P., and Tripathi, R. (2020b). A distributed framework for detecting ddos attacks in smart contract-based blockchain-iot systems by leveraging fog computing. *Transactions on Emerging Telecommunications Technologies*, n/a(n/a):e4112.

[Kumar et al. 2021b] Kumar, P., Tripathi, R., and P. Gupta, G. (2021b). P2idf: A privacy-preserving based intrusion detection framework for software defined internet of things-fog (sdiot-fog). In *Adjunct Proceedings of the 2021 International Conference on Distributed Computing and Networking*, ICDCN '21, page 37–42, New York, NY, USA. Association for Computing Machinery.

[Kumar et al. 2022] Kumar, R., Kumar, P., Tripathi, R., Gupta, G. P., Garg, S., and Hassan, M. M. (2022). A distributed intrusion detection system to detect ddos attacks in blockchain-enabled iot network. *Journal of Parallel and Distributed Computing*, 164:55–68.

[Kumar and Tripathi 2021] Kumar, R. and Tripathi, R. (2021). Dbtp2sf: a deep blockchain-based trustworthy privacy-preserving secured framework in industrial internet of things systems. *Transactions on Emerging Telecommunications Technologies*, 32(4):e4222.

[Labiod et al. 2022] Labiod, Y., Amara Korba, A., and Ghoualmi, N. (2022). Fog computing-based intrusion detection architecture to protect iot networks. *Wireless Personal Communications*, 125(1):231–259.

[Lalouani and Younis 2021] Lalouani, W. and Younis, M. (2021). Robust distributed intrusion detection system for edge of things. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 01–06.

[Lawal et al. 2020] Lawal, M. A., Shaikh, R. A., and Hassan, S. R. (2020). An anomaly mitigation framework for iot using fog computing. *Electronics*, 9(10).

[Lawal et al. 2021] Lawal, M. A., Shaikh, R. A., and Hassan, S. R. (2021). A ddos attack mitigation framework for iot networks using fog computing. *Procedia Computer Science*, 182:13–20. Learning and Technology Conference 2020; Beyond 5G: Paving the way for 6G.

[Li et al. 2021] Li, W., Au, M. H., and Wang, Y. (2021). A fog-based collaborative intrusion detection framework for smart grid. *International Journal of Network Management*, 31(2):e2107.

[Liu and Lang 2019] Liu, H. and Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, 9(20).

[Marín-Tordera et al. 2017] Marín-Tordera, E., Masip-Bruin, X., García-Almiñana, J., Jukan, A., Ren, G.-J., and Zhu, J. (2017). Do we all really know what a fog node is? current trends towards an open definition. *Computer Communications*, 109:117–130.

[Meidan et al. 2018] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. (2018). N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22.

[Meir and Rätsch 2003] Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. In *Advanced lectures on machine learning*, pages 118–183. Springer.

[Mell et al. 2011] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

[Miorandi et al. 2012] Miorandi, D., Sicari, S., De Pellegrini, F., and Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad hoc networks*, 10(7):1497–1516.

[Mitchell and Chen 2014] Mitchell, R. and Chen, I.-R. (2014). A survey of intrusion detection techniques for cyber-physical systems. *ACM Comput. Surv.*, 46(4).

[Mitchell 1997] Mitchell, T. M. (1997). Machine learning. *McGraw Hill Science/Engineering/Math*, page 432.

[Muhammad et al. 2015] Muhammad, F., Anjum, W., and Mazhar, K. S. (2015). A critical analysis on the security concerns of internet of things (iot). *International Journal of Computer Applications*, 111(7).

[Mukherjee et al. 1994] Mukherjee, B., Heberlein, L. T., and Levitt, K. N. (1994). Network intrusion detection. *IEEE network*, 8(3):26–41.

[Navas et al. 2018] Navas, R. E., Le Bouder, H., Cuppens, N., Cuppens, F., and Papadopoulos, G. Z. (2018). Do not trust your neighbors! a small iot platform illustrating a man-in-the-middle attack. In *International Conference on Ad-Hoc Networks and Wireless*, pages 120–125. Springer.

[Neshenko et al. 2019] Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., and Ghani, N. (2019). Demystifying iot security: an exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys & Tutorials*, 21(3):2702–2733.

[Nguyen et al. 2019] Nguyen, T. G., Phan, T. V., Nguyen, B. T., So-In, C., Baig, Z. A., and Sanguanpong, S. (2019). Search: A collaborative and intelligent nids architecture for sdn-based cloud iot networks. *IEEE Access*, 7:107678–107694.

[Ni et al. 2018] Ni, J., Zhang, K., Lin, X., and Shen, X. (2018). Securing fog computing for internet of things applications: Challenges and solutions. *IEEE Communications Surveys & Tutorials*.

[Northcutt et al. 2001] Northcutt, S., Cooper, M., Fearnow, M., and Frederick, K. (2001). *Intrusion Signatures and Analysis*. New Riders, New Jersey.

[Patel et al. 2010] Patel, A., Qassim, Q., and Wills, C. (2010). A survey of intrusion detection and prevention systems. *Information Management & Computer Security*.

[Ravi and Shalinie 2020] Ravi, N. and Shalinie, S. M. (2020). Semi-supervised learning based security to detect and mitigate intrusions in iot network. *IEEE Internet of Things Journal*, pages 1–1.

[Rey et al. 2022] Rey, V., Sánchez Sánchez, P. M., Huertas Celdrán, A., and Bovet, G. (2022). Federated learning for malware detection in iot devices. *Computer Networks*, 204:108693.

[Rokach 2016] Rokach, L. (2016). Decision forest: Twenty years of research. *Information Fusion*, 27:111 – 125.

[Roman et al. 2018] Roman, R., Lopez, J., and Mambo, M. (2018). Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698.

[Russell and Norvig 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition.

[Russell and Norvig 2010] Russell, S. J. and Norvig, P. (2010). *Artificial intelligence: A Modern Approach*. Pearson Education, Inc.

[Sahar et al. 2021] Sahar, N., Mishra, R., and Kalam, S. (2021). Deep learning approach-based network intrusion detection system for fog-assisted iot. In *Proceedings of international conference on big data, machine learning and their applications*, pages 39–50. Springer.

[Sarhan et al. 2021] Sarhan, M., Layeghy, S., Moustafa, N., and Portmann, M. (2021). Towards a standard feature set of nids datasets. *arXiv preprint arXiv:2101.11315*.

[Satyanarayanan 2015] Satyanarayanan, M. (2015). A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets. *GetMobile: Mobile Computing and Communications*, 18(4):19–23.

[Schapire 1990] Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, 5(2):197–227.

[Sharafaldin et al. 2018] Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*, pages 108–116.

[Sicari et al. 2015] Sicari, S., Rizzardi, A., Grieco, L., and Coen-Porisini, A. (2015). Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146 – 164.

[Souza et al. 2022a] Souza, C. A., Westphall, C. B., and Machado, R. B. (2022a). Two-step ensemble approach for intrusion detection and identification in iot and fog computing environments. *Computers & Electrical Engineering*, 98:107694.

[Souza et al. 2022b] Souza, C. A., Westphall, C. B., Machado, R. B., Loffi, L., Westphall, C. M., and Geronimo, G. A. (2022b). Intrusion detection and prevention in fog based iot environments: A systematic literature review. *Computer Networks*, page 109154.

[Souza et al. 2020] Souza, C. A., Westphall, C. B., Machado, R. B., Sobral, J. B. M., and dos Santos Vieira, G. (2020). Hybrid approach to intrusion detection in fog-based iot environments. *Computer Networks*, 180:107417.

[Sundhari 2011] Sundhari, S. S. (2011). A knowledge discovery using decision tree by gini coefficient. In *2011 International Conference on Business, Engineering and Industrial Applications*, pages 232–235.

[Takabi et al. 2010] Takabi, H., Joshi, J. B. D., and Ahn, G. (2010). Security and privacy challenges in cloud computing environments. *IEEE Security Privacy*, 8(6):24–31.

[Tanaka and Yamaguchi 2017] Tanaka, H. and Yamaguchi, S. (2017). On modeling and simulation of the behavior of iot malwares mirai and hajime. In *2017 IEEE International Symposium on Consumer Electronics (ISCE)*, pages 56–60.

[Tavallaee et al. 2009] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE.

[Traganitis et al. 2018] Traganitis, P. A., Pagès-Zamora, A., and Giannakis, G. B. (2018). Blind multiclass ensemble classification. *IEEE Transactions on Signal Processing*, 66(18):4737–4752.

[Ullah and Mahmoud 2020a] Ullah, I. and Mahmoud, Q. H. (2020a). A scheme for generating a dataset for anomalous activity detection in iot networks. In *Canadian Conference on Artificial Intelligence*, pages 508–520. Springer.

[Ullah and Mahmoud 2020b] Ullah, I. and Mahmoud, Q. H. (2020b). A scheme for generating a dataset for anomalous activity detection in IoT networks. In *Advances in Artificial Intelligence*, pages 508–520. Springer International Publishing.

[Vaccari et al. 2020] Vaccari, I., Chiola, G., Aiello, M., Mongelli, M., and Cambiaso, E. (2020). Mqttset, a new dataset for machine learning techniques on mqtt. *Sensors*, 20(22).

[Verma and Ranga 2019] Verma, A. and Ranga, V. (2019). Evaluation of network intrusion detection systems for rpl based 6lowpan networks in iot. *Wireless Personal Communications*, 108(3):1571–1594.

[Verma and Ranga 2020] Verma, A. and Ranga, V. (2020). Machine learning based intrusion detection systems for iot applications. *Wireless Personal Communications*, 111(4):2287–2310.

[Yan et al. 2016] Yan, Q., Yu, F. R., Gong, Q., and Li, J. (2016). Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys Tutorials*, 18(1):602–622.

[Yaseen et al. 2017] Yaseen, Q., Albalas, F., Jararwah, Y., and Al-Ayyoub, M. (2017). Leveraging fog computing and software defined systems for selective forwarding attacks detection in mobile wireless sensor networks. *Transactions on Emerging Telecommunications Technologies*, 29(4):e3183.

[Zarpelão et al. 2017] Zarpelão, B. B., Miani, R. S., Kawakani, C. T., and [de Alvarenga], S. C. (2017). A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, 84:25 – 37.