Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science in Engineering - Robotics
30.0 credits

ANOMALY DETECTION FOR NETWORK TRAFFIC IN A RESOURCE CONSTRAINED ENVIRONMENT

Gaia Ingletto
gia18002@student.mdh.se

Pontus Lidholm
plm18001@student.mdh.se

Examiner: Ning Xiong
Mälardalen University, Västerås, Sweden

Supervisor(s): Miguel León Ortiz and Tijana Markovic
Mälardalen University, Västerås, Sweden

Company Supervisor(s): Andreas Egeberg and Per Erik Strandberg
Westermo Network Technologies AB, Västerås, Sweden

2023

Abstract

Networks connected to the internet are under a constant threat of attacks. To protect against such threats, new techniques utilising already connected hardware have in this thesis been proven to be a viable solution. By equipping network switches with lightweight machine learning models, such as, Decision Tree and Random Forest, no additional devices are needed to be installed on the network. When an attack is detected, the device may notify or take direct actions on the network to protect vulnerable systems. By utilising container software on Westermo's devices, a model has been integrated, limiting its computational resources. Such a system, and its building blocks, are what this thesis has researched and implemented. The system has been validated using multiple different models using a range of parameters. These models have been trained offline on datasets with pre-recorded attacks. The recordings are converted into flows, decreasing dataset size and increasing information density. These flows contain features corresponding to information about the packets and statistics about the flows. During training, a subset of features was selected using a Genetic Algorithm, decreasing the time for processing each packet. After the models have been trained, they are converted to C code, which runs on a network switch. These models are verified online, using a simulated factory, launching different attacks on the network. Results show that the hardware is sufficient for smaller models and that the system is capable of detecting certain types of attacks.

Acknowledgements

The authors would like to express their deepest gratitude to the supervisors, Miguel León Ortiz and Tijana Markovic, for all the invaluable advice, detailed feedback, and proofreading throughout the thesis. Furthermore, this endeavour would not have been possible without Dr. Per Erik Strandberg and Andreas Egeberg who supplied their support, time and vast knowledge throughout the project. Thanks should also go to Westermo, for the warm welcome, lovely atmosphere, colleagues, and sublime coffee.

Some emojis were designed by OpenMoji – The open-source emoji and icon project. License: CC BY-SA 4.0

Some figures were created in Draw IO - A open-source, free to use software for any purpose.

Figure 10 was created by P. E. Strandberg, D. Söderman, A. Dehlaghi-Ghadim, M. Leon Ortiz, T. Markovic, S. Punnekkat, M. Helali Moghadam, and D. Buffoni. (2023). The figure was retrieved from the Westermo network traffic data set, https://github.com/westermo. License: CC BY 4.0

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ARP	Address Resolution Protocol
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DNS	Domain Name Server
DoS	Denial of Service
DT	Decision Tree
GA	Genetic Algorithm
IDS	Intrusion Detection System
IG	Information Gain
IoT	Internet of Things
IPFIX	IP Flow Information Export
KNN	K-Nearest-Neighbour
LAN	Local-area network
LIDS	Lightweight Intrusion Detection System
LXC	Linux Container
MITM	Man-in-the-Middle
ML	Machine Learning
NIC	Network Interface Card
PoE	Power over Ethernet
R2L	Remote-to-Local
RF	Random Forest
SSH	Secure Shell
SFP	Small Form-factor Pluggable
SVM	Support Vector Machine
TCP	Transmission Control Protocol
U2R	User-to-Root
VM	Virtual Machine
XGboost	Extreme Gradient Boosting

Contents

List of Figures

List of Tables

1. Introduction

The Internet is growing at an exponential pace and with this, advanced security threats emerge. Cyberattacks are evolving and exploiting the weakest parts of the Internet's defences. One such weak spot is the Internet of Things (IoT) branch, which consists of simple devices, e.g. smart lamps, coffee machines, and door locks, which are common targets of attacks. The IoT business consists of tens of billions of devices and is steadily growing [1]. To ensure that networks with an increasing number of vulnerable devices stay secure, defence mechanisms must adapt quickly to these new threats [2], [3]. With this and an increased reliance on our networks, improving cybersecurity and protecting critical systems is essential. For an organisation, cyberattacks can lead to lost revenue and damage to its reputation. Cyberattacks can also be dangerous to societies if for example, the attacks cause power outages or if national security breaches are revealed [4].

The complexity of cyber threats is rising, but at the same time, devices connected to the Internet have become more powerful. This may leave a surplus of computational power such that it could be possible for more advanced defensive algorithms to be hosted. A subset of the possible defensive algorithms is based on Artificial Intelligence (AI). The success and accuracy of AI algorithms are linked to the computational power of the host device [5]. Therefore this performance increase in devices has opened a market for even smaller units such as routers and switches to host an AI. Such systems could increase cybersecurity and lower the impact of cyberattacks when they occur. But an AI operating in a network should not be limited to only detecting cyber threats, but could also be used to detect device malfunctions or configuration mistakes. For example, an AI could notice if a device in an industrial environment malfunctions and starts to send irrelevant packets [6]. With these new opportunities, it is appropriate to research how such a method should be implemented for optimal performance. This is where this thesis, in collaboration with Westermo Network Technologies AB (Westermo)[1], aim to increase cybersecurity in an industrial control network. This has been explored by developing a Machine Learning (ML) model which operates from the inside of a container, running on a layer 3 switch. The container runtime chosen in this thesis is Linux Container (LXC), because of support from Westermo's devices. Operating from within the container, the ML algorithm monitors packets flowing through the switch and classifies if these are part of an anomaly or normal operation of the network [7].

1.1 Problem Formulation

This thesis proposes to answer questions regarding the detection of anomalies, using ML. The ML model will operate from a layer 3 switch that is connected to a network. From previous studies in this field, Random Forest (RF) has proven to give high accuracy in network anomaly detection, while also being lightweight in terms of computational resources. This suggests RF to be a suitable algorithm for the proposed system. The switches that the system will operate from are provided by Westermo. Based on the problem described in the Introduction, the following questions are stated.

1. Is it possible, and if so, how well does random forest perform anomaly detection in a resource-constrained setting?
2. Which actions are required in order to implement an anomaly detection system for network traffic on a switch with limited computational resources?
3. What information, features, in network traffic, help to improve the performance of the network traffic classification system?

1.2 Thesis outline

This thesis is organised in the following layout: Section 2. states the relevant background that is needed for a complete understanding of this thesis, i.e., network anomalies, and ML algorithms. Section 3. brings up related works, similar projects that have been done before. Section 4. states the research method that has been followed during the thesis. Section 5. contains a detailed description

[1]https://www.westermo.se

of the implementation. Section 6. contains the results that have been produced throughout the project. Section 7. discusses the results of the thesis and interprets the significance and meaning of the results. Section 8. concludes the thesis work. Section 9. suggests steps for future work, where the authors propose potential additions for future projects.

2. Background

In this section, the relevant background for the thesis is presented. The theory is required for a deeper understanding of the subject of cyber threats and common risks for a network. Then, network flows and feature selection is described. Later in the section, the ML algorithms used are described. The section is concluded with information about intrusion detection datasets, containers and the host device.

2.1 Threats

Cyberattacks have been on the rise, and the complexity of their techniques is increasing. Additionally, new advanced tools require less knowledge of their targets. In Table 1, a multitude of common attacks is listed. Many of these may cause a significant disruption in the operation of a network [8]–[10]. There exist numerous methods for hackers to achieve their goals, no matter their objectives. The methods may be divided into two categories, either passive or active. A passive attack only monitors the target without any interference or modification to collect information, while an active attack causes direct damage.

Attack	Description	Type
Denial of Service	A DoS attack overloads a network to prevent others access to the service.	Active
Distributed DoS	A version of DoS where multiple machines are used for a more powerful attack.	Active
Code injection	Running unauthorised code by injecting code into a user interface.	Active
Spoofing	When a device or node masks its identity for something else.	Active
Modification	Increases communication delay by modifying the routing of a network.	Active
Wormhole	Exploiting tunnelling of packets between two points in a network.	Active
Sinkhole	Modifying or dropping packets to prevent correct information from arriving.	Active
Cryptojacking	Causing a system to mine cryptocurrency in the background.	Active
Remote-to-Local (R2L)	Unauthorised access from a remote machine, e.g. password guessing.	Active
User-to-Root (U2R)	Unauthorised access to local root privileges, e.g. buffer overflow.	Active
Password cracking	Using techniques such as brute force, dictionary or phishing.	Active
Buffer overflow	Exceeding an inputs capacity to write in another piece of memory.	Active
Probing	Surveillance or other information gatherings, e.g. port scanning.	Passive
Traffic analysis	Collecting data about communication transceiving between two or more nodes.	Passive
Insider Threats	Someone inside a network disturbing or listening in on a network.	Both
Man in the Middle	An attacker may hijack information between two communicating parties.	Both

Table 1: Common network attacks.

2.1.1 DoS and DDoS

A Denial of Service (DoS) attack may target either the provider of a service or the user, to deny access to a service. This is often accomplished by exhausting the resources of either party. This may involve resources such as Central Processing Unit (CPU), storage, or network on the server as well as network, router, or bandwidth of the client. Such exhaustion may be achieved by sending a large number of packets to a single address. The packets could use different protocols and flags to trigger a response. Other methods exist, depending on the receiver. For example, if a web server is the target of the attack, a large number of requests of the site may be transmitted to overload the server. Today, because of the increased capabilities of devices, a majority of DoS attacks are Distributed Denial of Service (DDoS) attacks. What makes DDoS different from DoS is that instead of only one device trying to overload a service, multiple devices coordinate an attack simultaneously. The recruitment of these devices is often achieved by using other types of attacks. This kind of attack is often harder to detect and trace [11], [12]. An example of a DDoS attack is shown in Figure 1.

2.1.2 Spoofing

Spoofing is the act of disguising a communication from an unknown source as being a known trusted source. There are multiple targets of information that are possible to spoof. Transmission Con-

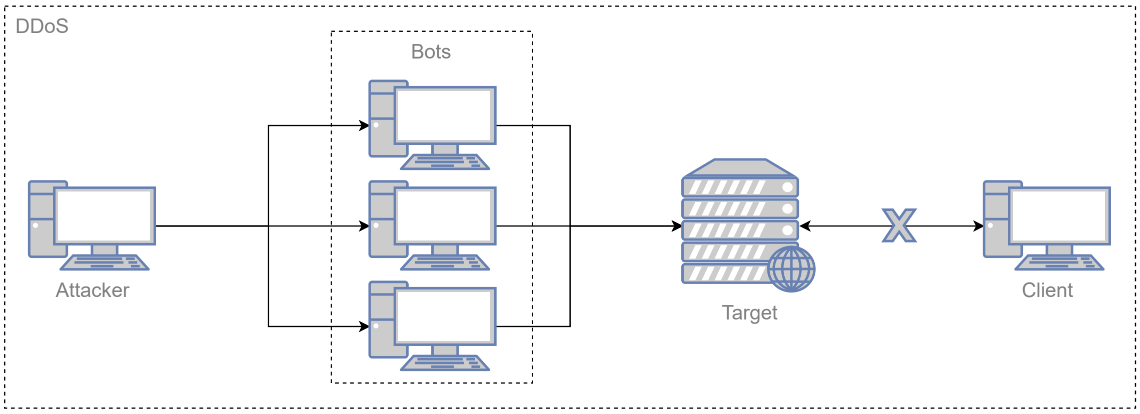


Figure 1: A representation of a DDoS attack. In this case, the attacker has control of a set of bots, that is commanded to transmit a large number of packages towards the target.

trol Protocol (TCP) spoofing is one example, which builds upon modifying the attached transmit address that is included in the header. Such that the assumed transmitter's identity is incorrect. This makes the receiving computer respond incorrectly. A similar technique is Domain Name Server (DNS) spoofing where the attacker has control over the information about devices' corresponding addresses. A more powerful version of spoofing, called web spoofing, is where the attacker has control over all the data that is transmitted between a client and the spoofed web. This is accomplished by providing a copy of the requested, real website, and listening and possibly modifying the communication to the copied site [13], [14].

2.1.3 Port Scanning

Port scanning does no harm on its own and is more of a recognisance technique than an attack. It is often used to locate potential targets, which is why the technique is still a threat. A port scan could be executed both vertically and horizontally. In a vertical attack, one host is scanned on all ports of interest. While in a horizontal scan, multiple hosts are targeted on a specific port. The former is often used when there is a specific target, while the latter is more relevant when trying to exploit a common weakness. The complexity ranges from a more simple brute force attack to more complex methods that focus on stealth to avoid detection and prevention. A brute force attack tries to establish a connection to each port in a range. Depending on the connection, the scanner assumes the state of the port. Stealth attacks avoid establishing a connection which makes them a little harder to detect. These attacks vary in the type of protocol and flags that are used [15], [16]. An example of a port scan is shown in Figure 2.

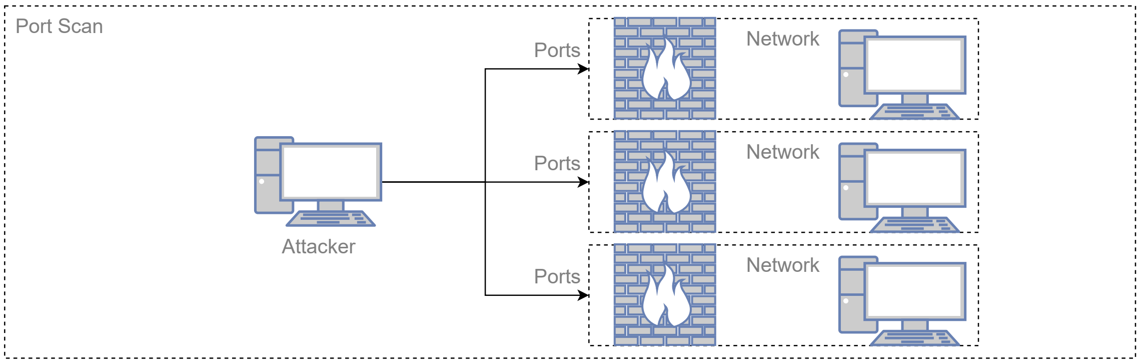


Figure 2: An example of a port scan attack. An attacker sends packets to multiple separate networks while looking for entry points.

2.1.4 Man in the Middle

Man-in-the-Middle (MITM) is mostly suited for an attack on a Local-area network (LAN). There exist multiple ways of initialising this kind of attack e.g. Address Resolution Protocol (ARP) poisoning. Each device in a network has an ARP table. The ARP table is a method used to store discovered pairs of MAC and IP addresses. When this table is poisoned, MAC and IP pairs may not match such that packets are sent to the wrong devices. Regardless of technique, the attacker establishes a network connection such that all communication between the two parties is transported via the attacker. This results in full access to unencrypted information and possible modification to the communication [17]. An example of a MITM attack is shown in Figure 3.

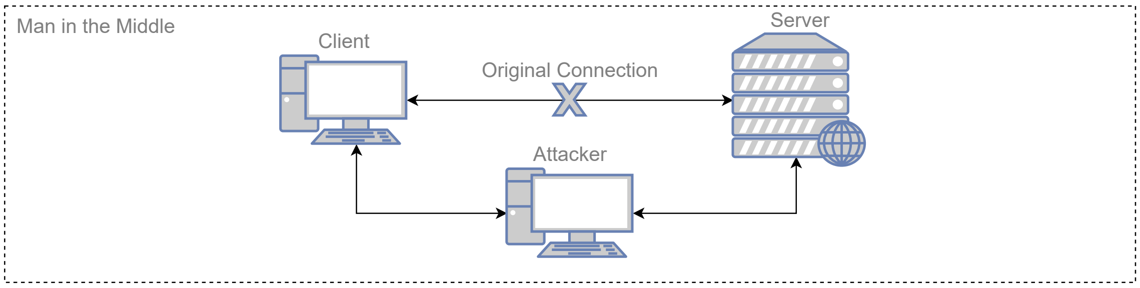


Figure 3: A MITM attack shown. The attacker has hijacked the connection between a client and a web server.

2.2 IP Flow Information Export

The IP Flow Information Export (IPFIX) is a protocol designed for network administration and monitoring. The basis of the standard originates from Cisco Net-Flow [18]. IPFIX makes it possible to collect network traffic going through an interface. Its main purpose is to compile a large amount of network data in the form of individual packets into a more compact format, namely, flows [19]–[21].

When analysing traffic and sorting packets, there are a few features that determine if a packet is part of a flow. These are listed in Table 2 with a short description.

Attribute	Description
src ip	Source address of the flow
dst ip	Destination address of the flow
dport	Destination port of the flow
protocol	Protocol (UDP, TCP, ARP, ...)

Table 2: The basis features for IPFIX.

By grouping packets into flows, administrators can collect vital information on larger networks without collecting and storing data about each packet. This saves storage space without discarding key information. The IPFIX standard is flexible in what type of data, and features are exported from the device. This enables specifying the exported data to fit the configuration of the network, requirements from the analyses, and the capacity of the storage. In most cases, only a few features are exported, but all possible features are listed at [22]. An example of a setup implementing flow export is shown in Figure 4. This setup includes a flow server, connected to a small network that includes clients, a router, and an exporting device.

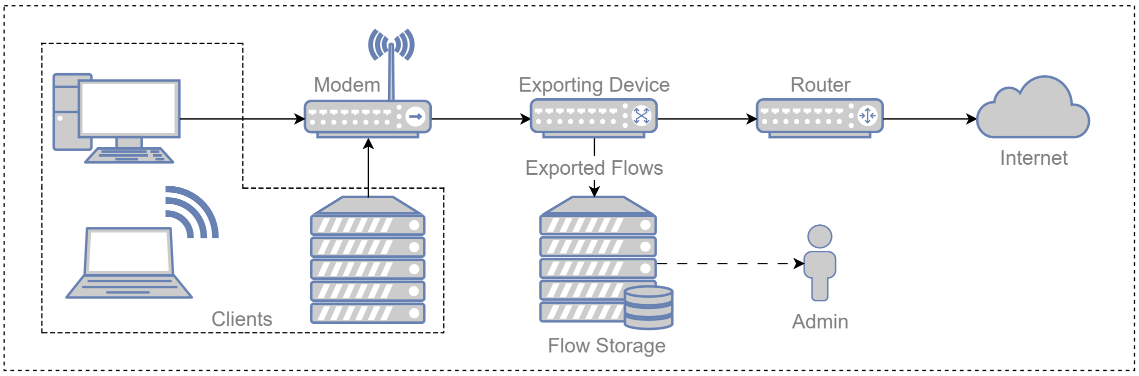


Figure 4: An example setup of a network that supports the export of flows. To the left is a set of network clients, that is connected either wired or wireless to a modem. Between the modem and the router is a device connected so that all data passes through the device before reaching the router and later the internet. This allows the device to process all packets and converts them into flows. This data is later exported to a server for storage and processing.

2.3 Feature selection

When packets are grouped into flows, the comprehensive data that all packets contain are converted into features. Because of the considerable amount of packets that may travel through a network device, it is essential to constrain the number of features. This is exceptionally important due to the limited amount of computational power of a network device. Therefore, only the most vital features should be calculated. Feature selection is a comprehensive subject, and there exists a large number of different strategies. Some of the more common methods are complete search, heuristic, and random. In the complete search category, there are algorithms like breath-first and depth-first. These are exhaustive methods, extensively trying all the methods until a stop criterion is met. While in the heuristic category, more efficient algorithms are found that settle on a solution that may not be the best possible, but instead is good enough [23]. This thesis focuses on the last category, random, or more specifically, a Genetic Algorithm (GA), to deduce which features are more important.

2.4 Genetic algorithm

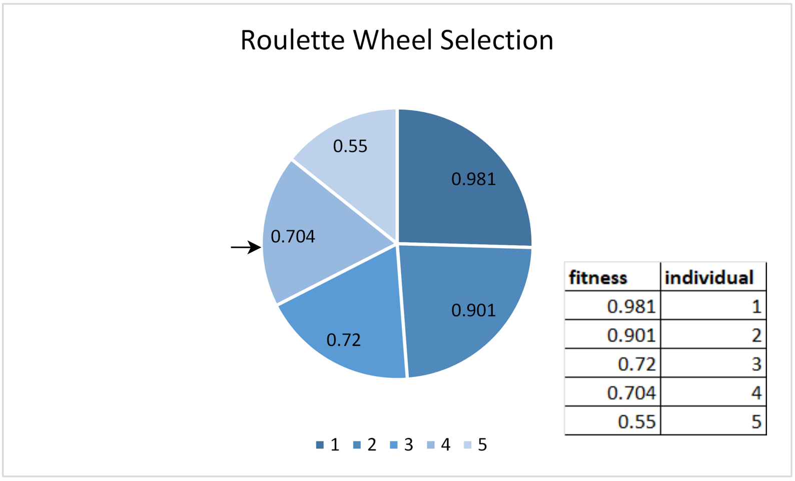
GAs are inspired by natural selection and other biological processes such as mutation and crossover. A GA can be used to solve optimisation problems and search problems. The pseudo-code for a traditional GA can be found in Algorithm 1.

Algorithm 1 GeneticAlgorithm(population_size)

```
population = InitializePopulation()
CalculateFitness(population)
while True do
    if Termination criteria is satisfied then
        Return best_individual
    else
        parents = SelectParents(population)
        offspring = Crossover(parents)
        Mutate(offspring)
        CalculateFitness(offspring)
        new_population = Replace(population,offspring)
    end if
end while
```

The first step in a GA is to initialise all the individuals in a population. The individuals are different depending on how they are represented. The individuals can be for example binary arrays, arrays with real numbers, or arrays with integers representing an order. The individuals can then be initialised randomly or individuals from a previous search can be used. After a population is created, the fitness value of each individual in the population is calculated. How the fitness values are calculated depends on the application of the GA.

Once all the fitness values are calculated, it is time to select the parents if the termination criterion is not met. The termination criterion can be a certain number of generations that have passed or generations without any improvement. There exist various methods to do this, but in all methods, the selection depends on the fitness values. The methods that are described here are Roulette selection and Tournament selection. In Roulette selection, each individual gets a probability of being selected. This probability is proportional to the fitness value of the individual. Equation 1 shows how to calculate the probability of an individual being selected as a parent. Figure 5 shows the principles of Roulette selection.



fitness	individual
0.981	1
0.901	2
0.72	3
0.704	4
0.55	5

Figure 5: An example showing the principles of Roulette selection. The higher the fitness value of an individual, the greater its area in the roulette wheel and the more probable for that individual to be selected.

$$p_x = \frac{f_x}{\Sigma_{i=1}^{n} f_i} \tag{1}$$

1: f_x is the fitness value of individual x, n is the number of individuals in the population and p_x is the probability of individual x to be selected as a parent.

In Tournament selection, k individuals are randomly selected for the tournament. From the tournament, the best individuals are selected as parents. After selecting parents a crossover is performed in order to create offspring. How the offspring are created depends on the representation of the individuals. If binary representation is used one or multiple crossover points are selected. An example with one crossover point can be seen in Figure 6. If a real number representation is used, the crossover can be performed using arithmetic recombination or BLX-α [24]. To create one offspring using arithmetic recombination the mean values of the parents are used. Element i of the offspring is equal to the mean of element i of parent1 and element i of parent2. The algorithm BLX-α can also be used in a real number representation. See Figure 7 for a visual explanation of the algorithm.

After creating offspring they can be mutated. How often the offspring are mutated depends on the mutation rate. There are different ways to mutate an offspring. If a binary representation is

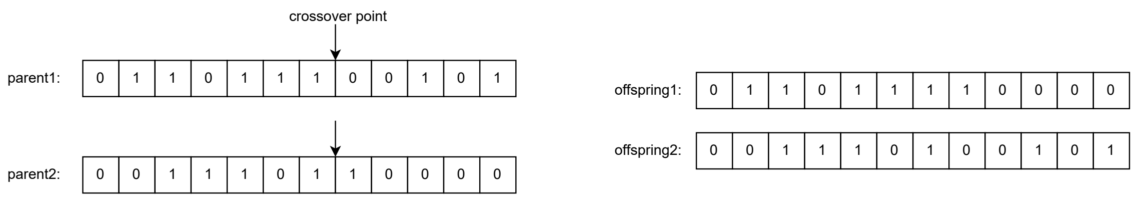

crossover point
parent1: 0 1 1 0 1 1 1 0 0 1 0 1
parent2: 0 0 1 1 1 0 1 1 0 0 0 0
offspring1: 0 1 1 0 1 1 1 1 0 0 0 0
offspring2: 0 0 1 1 1 0 1 0 0 1 0 1

Figure 6: Single point crossover using binary representation. Offspring1 gets its first seven elements from the first seven elements of parent1 and the last four from the last four of parent2. The same goes for offspring2 but the first elements from parent2 and the last elements from parent1.

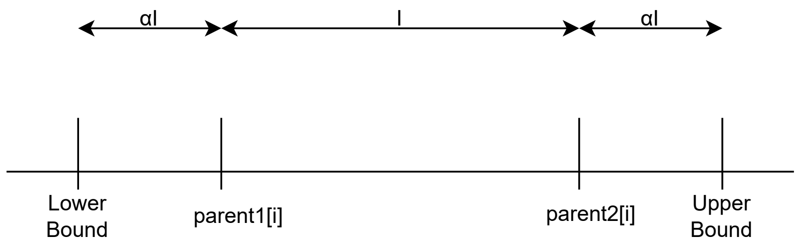


Figure 7: Element i of each offspring is taken randomly from the interval between the upper and lower bound.

used, one or multiple bits can be flipped. With a real representation, the normal density function is used such that offspring1[i] = offspring1[i] + N(μ,σ)

After mutation and calculating the fitness of the offspring, the offspring are ready to replace the population. In a generational GA, offspring replace the entire population or a certain number of best individuals are kept. In a stationary GA, only the worst individuals are replaced.

2.5 Supervised learning

In supervised learning, the goal is to learn a mapping function that maps an input to an output by observing input and output pairs. The outputs that are observed are called labels. For example, in anomaly detection, the labels can be either normal or attack. The dataset contains cases which consist of a certain number of attributes and one output. Supervised learning problems can be divided into two types: classification and regression. In classification, the output is a class. In regression, the output is a continuous-valued output. There exist multiple ML algorithms that learn in a supervised manner. The following subsections will describe a selection of these algorithms, with a focus on their classification problem-solving capabilities [25].

2.5.1 Decision trees

A Decision Tree (DT) can be used for classification. The DT model is used to predict the target value by learning decision rules deduced from the data attributes. A DT begins with a root with branches leading to either an internal node or to a leaf. Each leaf contains a class label. Each internal node splits the instance space into subspaces according to a splitting function and the corresponding branches to that node are the outcome of the splitting function. In most cases, a node splits the instance space according to the value of one attribute. At every node in a DT, the value of one of the parameters, the branching variable, is compared against a threshold. The branching variable and the threshold are parameters that the algorithm learns [26]. The DT algorithm searches for the attribute which gives the best split. There are various ways to search for this attribute. The most common way in literature is the impurity-based criteria [27].

Impurity-based criteria: If $P = (p_1, ..., p_k)$ is the distribution of a random variable with k discrete values, where k is the number of classes. The function $\phi : [0, 1] \to \mathbb{R}$ defines an impurity measure if the following conditions are satisfied:

1. $\phi(P) \geq 0$
2. If $p_i = 1$ for some i then $\phi(P)$ is minimum
3. If $\forall i, 1 \geq i \geq k, p_i = \frac{1}{k}$ then $\phi(P)$ is maximum
4. $\phi(P)$ is symmetric with respect to $p_1, ...p_k$
5. $\phi(P)$ differentiable everywhere in its range.

For example, one impurity measure is Information Gain (IG). The DT algorithm compares every possible split and takes the one that maximises the IG. To calculate IG, the entropy needs to be calculated first. Entropy can be described as a measure of disorder. In general, the entropy of the nodes decreased as the tree is traversed. This means that attributes considered at the beginning of a tree are in general more important as these splits can lead to higher IG. IG specifies the impurity in class elements and is defined as the difference between entropy before and after a split:

$$Entropy(current\_node) = \Sigma_{i=1}^{k}(-P_i * log_2(P_i)) \tag{2}$$

Where P_i is the proportion of class i after a split in the current node and k is the number of classes.

$$IG = Entropy(parent) - \Sigma_{i=1}^{k} v_i * Entropy(child) \tag{3}$$

Where $v_i \in [0, 1]$ is the fraction of data affected. A DT is constructed by choosing the attribute and threshold that maximises the IG. A simple way to choose the thresholds is to first order the values of the attributes in numerical order. Then consider only the average values of the numeric attributes where the value of the target switches [27]. Figure 8 shows how the IG is calculated.

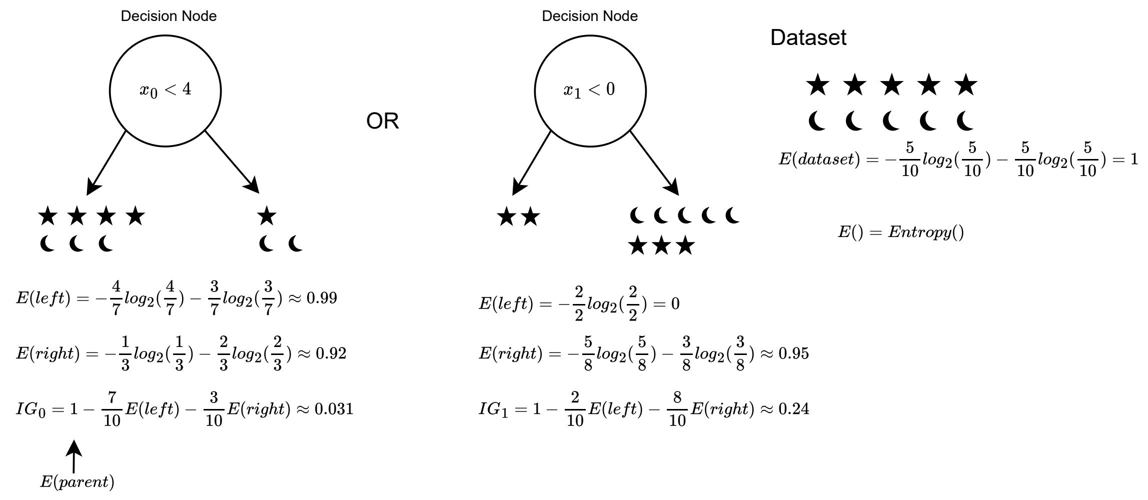

Decision Node
$x_0 < 4$
OR
Decision Node
$x_1 < 0$
Dataset
$E(dataset) = -\frac{5}{10}log_2(\frac{5}{10}) - \frac{5}{10}log_2(\frac{5}{10}) = 1$
$E() = Entropy()$
$E(left) = -\frac{4}{7}log_2(\frac{4}{7}) - \frac{3}{7}log_2(\frac{3}{7}) \approx 0.99$
$E(right) = -\frac{1}{3}log_2(\frac{1}{3}) - \frac{2}{3}log_2(\frac{2}{3}) \approx 0.92$
$IG_0 = 1 - \frac{7}{10}E(left) - \frac{3}{10}E(right) \approx 0.031$
$E(parent)$
$E(left) = -\frac{2}{2}log_2(\frac{2}{2}) = 0$
$E(right) = -\frac{5}{8}log_2(\frac{5}{8}) - \frac{3}{8}log_2(\frac{3}{8}) \approx 0.95$
$IG_1 = 1 - \frac{2}{10}E(left) - \frac{8}{10}E(right) \approx 0.24$

Figure 8: Example showing how to calculate the IG. In this example, k = 2 and when $p_i = 1/k$ the entropy is 1, which is the maximum. This is a binary classification to determine if an input is a star or a moon. Here the attribute x_1 and the threshold 0 will be chosen as $IG_1 > IG_0$. IG_1 is greater since performing the split, $x_1 < 0$ gives a more ordered set. The leaf to the left has only one class and this gives the minimum disorder.

If looking at Figure 8 and the equations for entropy and IG, it can be seen that the function for calculating IG satisfies the conditions of the impurity-based criteria.

The pseudo-code for growing a DT is based on the algorithm [27] and can be seen in Algorithm 2.

Algorithm 2 TreeGrow(S,A,y)

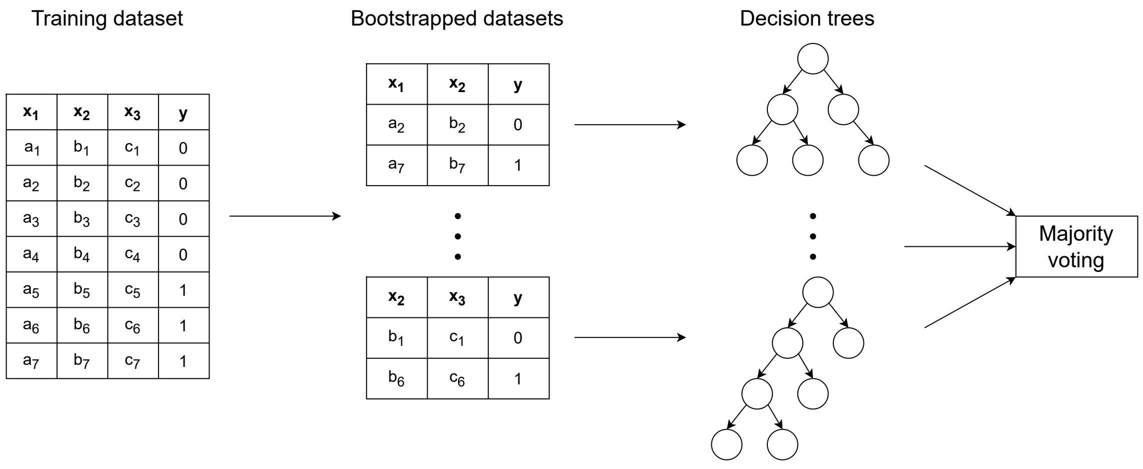
```
S = Training set
A = Input Attribute Set
y = Target
Create a new tree T containing only a root node
if one of the stopping criteria is satisfied then
    The root in T becomes a leaf with the most common value of y in S as a label
else
    Find a ∈ A that gives the best split of S
    i = 1
    for each possible value of a $(v_1, ..., v_n)$ do
        S = the subset of S where a has value $v_i$
        Remove a from A
        $Subtree_i$ = TreeGrow(S,A,y)
        Connect the root node of T to $Subtree_i$ with an edge labelled $v_i$
        Incremenet i
    end for
end if
Return T
```

In Algorithm 2, A is the attribute set of S. $(v_1, ..., v_n)$ are all the possible values that the attribute a can have and a is the attribute that best splits S according to some impurity based criteria. For example, if the attribute a is protocol, then the possible values of that attribute are all the protocols considered. Or if the attribute is the total number of bytes in a flow, then the values could, for example, be the number of bytes greater than 100 or the number of bytes less than or equal to 100. After splitting S a subtree is grown from each edge, $(v_1, ..., v_n)$. The new training set S for a $subtree_i$ is all the samples x, where the attribute a of x has value v_i. The new attribute set used as input to the recursive call to the function TreeGrow is the set of all attributes A, excluding a.

There exist multiple methods of growing a DT. Because finding the optimal tree is an NP-hard problem a heuristic strategy is primarily used. Some of these strategies build on a best-first or a divide-and-conquer basis. These recursive strategies will then grow a tree based on their respective rules. The strategies and their performance vary greatly depending on the data, which is why numerous methods exist [28]. The drawback with DTs is that they are highly sensitive to their training data which results in high variance. This can cause DTs to be unable to generalise [29].

2.5.2 Random Forest

A RF consists of multiple randomised DTs. In RF, each tree gets its own fraction of the dataset with entries taken randomly with replacement from the original dataset. Since it is with replacement the fractions of the dataset each tree gets can overlap. This process of creating new random datasets is called bootstrapping. Each DT will train on each of the bootstrapped datasets independently. When training a tree, not all the features are used, only a randomly selected subset. When using RF to classify the input each individual DT in the forest outputs one class prediction. The RF-model's prediction is the class which receives the most votes. Figure 9 shows an example illustrating this.



x_1	x_2	x_3	y
a_1	b_1	c_1	0
a_2	b_2	c_2	0
a_3	b_3	c_3	0
a_4	b_4	c_4	0
a_5	b_5	c_5	1
a_6	b_6	c_6	1
a_7	b_7	c_7	1

x_1	x_2	y
a_2	b_2	0
a_7	b_7	1

x_2	x_3	y
b_1	c_1	0
b_6	c_6	1

Figure 9: x_1, x_2, x_3 are the attributes of the dataset and y is the target, specifying which class the sample belongs to. After randomly sampling attributes and entries from the training set, the first bootstrapped dataset consists of attributes x_1, x_2 and entries 2 and 7. The last bootstrapped dataset is made up of attributes x_2, x_3 and entries 1 and 6. Each bootstrapped dataset points to a DT created by training on the corresponding bootstrapped dataset.

Random attribute selection makes the trees in the RF less correlated. Usually, the number of attributes to consider each time is the square root of the total number of attributes or the logarithm in base 2 of the total number of attributes [30].

A problem that can occur with DT or RF is overfitting. Overfitting occurs when the algorithm, instead of learning how to solve the task in general, adapts excessively to the training data. To avoid this, two common strategies are bagging and boosting. Bagging is the process of using bootstrapping and aggregation (majority voting). Bagging functions on the basis that the variance of a sample may be decreased without significantly altering the mean. It is achieved by dividing the sample into n random subgroups and calculating the mean of each group. This then makes up a new smaller dataset where the mean should not have changed significantly but the variance of the new set should be smaller, which should improve predictions. Whereas in boosting, instead of generating multiple sets and averaging in a parallel process, the data is sequentially updated to produce one final prediction. In this iterative process, each version will try to correct the last error [30], [31].

When generating a RF, the amount of trees is a large factor, as the processing time for the algorithm increases, but the improvement will converge. In a study conducted by Mayumi Oshiro et al. [32], 29 datasets were analysed using RF. The model increased in population, from 2 to 4096, doubling each step. The results show that the performance increased slowly up to 128 trees, after which improvement ceased. Suggesting that for most problems in resource-limited environments, a maximum of 128 trees should be grown.

2.6 Intrusion detection datasets

In this section, multiple intrusion detection datasets are described.

2.6.1 KDD99

The KDD99 [33] is a dataset that was made in the year 1999 for a competition (International Knowledge Discovery and Data Mining Tools Competition). The task of the competition was to distinguish between bad connections and normal connections. The dataset contains several different simulated intrusions. The data (raw TCP dump) were acquired over 9 weeks in a military network environment, simulating a U.S Air Force LAN. The LAN was operated as a true Air Force environment except for the injected attacks. The training data was recorded for 7 weeks and contains approximately 5 million flows. A flow is a sequence of packets, flowing from a source IP address to a destination IP address under some defined time interval and protocol. In the KDD99 dataset, each flow has a label and 42 features. The label is either a type of attack or normal. Each attack can be categorised as either DoS, R2L, U2R or probing. See Table 1 for a description of the attacks [33].

2.6.2 NSL-KDD

The NSL-KDD [34] dataset is a new version of KDD99. NSL stands for Network Security Laboratory. The entries in the training set are not redundant and there are no duplicates. This is to prevent the learning algorithms from being biased towards more frequent entries. Being biased towards more frequent records prevents the algorithms to learn less frequent records which can be more dangerous, such as U2R and R2L attacks. The NSL-KDD dataset contains a total of 0.13 million entries in the training set and 0.023 million entries in the test set. The instances are labelled as either normal or with the name of the attack. The dataset contains various types of attacks, including DDoS, DoS and port scan [34].

2.6.3 UNSW-NB15

Compared to KDD99 and NSL-KDD, the UNSW-NB15 [35] is a more modern dataset. UNSW-NB stands for University of New South Wales - Network Based. This dataset was created in 2015 and contains real normal traffic and synthesised network attacks. The dataset has 49 features, 9 different attack categories and 45 distinct IP addresses. The 49th feature is the label which is either normal or attack. The 48th feature is the attack category. The entire dataset contains 2.2 million entries labelled as normal and 0.32 million entries labelled as attack [35].

2.6.4 CIC-IDS-2017

The CIC-IDS-2017 (Canadian Institute for Cybersecurity - Intrusion Detection System 2017) dataset [36] was created in 2017 since most of the previous datasets are obsolete. Due to the evolution of network attack strategies, datasets need to be updated from time to time. The dataset was recorded for 5 days. On the first day, only normal traffic was recorded and the other days contain both attacks and normal traffic. The dataset has 84 features and 2.8 million entries. The attacks in the dataset can be categorised into DoS, DDoS, brute force, infiltration, port scan and Botnet among others [36].

2.6.5 Westermo network traffic dataset

The Westermo network traffic dataset[2] contains 1.8 million packets, that have been generated in a simulated factory. The data is recorded over a span of 90 minutes and the network consists of 12 devices. Multiple kinds of attacks have been recorded, as well as misconfigurations that are not necessarily malicious but can cause the switch not to function as intended. The attacks are as follows: login attempts using Secure Shell (SSH), port scan, MITM, and misconfiguration simulating human mistakes. Assigning an incorrect IP address is a common human mistake when configuring a device. These human mistakes include assigning an address that already exists on the network (duplication) or assigning an address with changed order. In the dataset, there are also non-malicious connections over SSH. Figure 10 shows the network topology Westermo used when recording data for their dataset.

2.7 Containers

Virtual Machine (VM) is a technique in which a physical computer hosts a guest system. The VM contains a complete operating system, but is isolated from the host device. In between the host and the VM is a hypervisor, which role is to manage all VMs and keep them isolated from each other. The hypervisor also allocates resources, such as memory, CPU, and disk space, to each VM at the host device.

[2]https://github.com/westermo/network-traffic-dataset

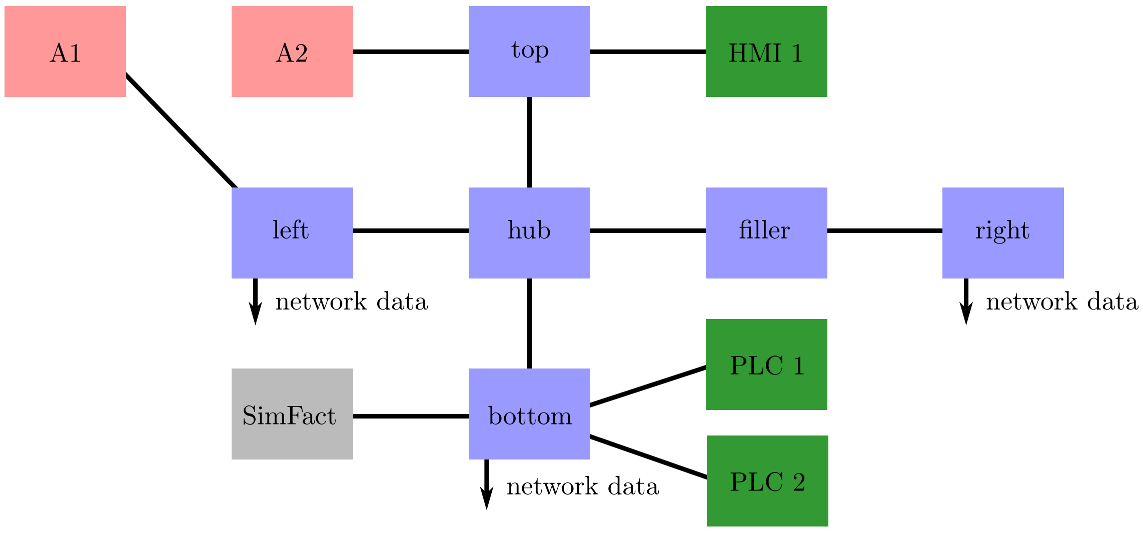


Figure 10: The network topology Westermo used when collecting data for the dataset. The data are packets coming into or out from either the left, right or bottom device. The attackers are A1 and A2 (Westermo 2023 [37]).

Compared to VMs containerisation is an approach of isolation, where Docker is available as a container platform. Docker[3] runs on the host operating system as a program and operates one or multiple containers. Docker isolates each container and therefore its content, the application, from the hardware. Each container also contains the settings and library needed, which makes them compatible with different machines independent of host systems.

Another approach is Kata Containers[4], which is an open-source container runtime that brings containers with larger isolation. Kata containers are a hybrid between the VMs and the containerisation of Docker. Kata Containers implements a lightweight virtual machine containing its own kernel that hosts multiple containers. This is implemented to isolate the containers from each other and the physical hardware without the performance impact of a VM.

A main feature of a container is that it should isolate the different programs running inside containers, and with the host operating system. A security problem is container escape, e.g. CVE-2019-5736, where it was possible to gain root access to the host from inside the container. This exploit defeats the main purpose of using a container and undermines the security of the system.

Containers are a type of virtualisation technology. Unlike virtual machines, containers have low usage of system resources, they do not emulate hardware and they share the same operating system as the host.

The container interface used in this thesis is LXC. This is because of the support on the devices that have been acquired, which builds upon a business decision of Westermo. LXC is an OS-level virtualisation technology that can simultaneously run multiple Linux systems (containers) in isolation on a single Linux kernel (LXC host). The building blocks for LXC are namespaces and control groups (Cgroups). Namespaces are a Linux kernel feature that can isolate processes from each other. Namespaces make partitions of the kernel resource such that one set of processes sees one set of resources and another set of processes sees another set of resources. Cgroups are also a feature of the Linux kernel. A Cgroup allows for the limitation of resource usage and prioritisation of resources [38].

[3]https://docs.docker.com/
[4]https://katacontainers.io/

2.8 Host - Device

The host device that is used for this thesis work is provided by Westermo. Westermo is a company that develops robust network devices that is to be deployed in harsh environments. Their devices are both layer 2 and layer 3 switches, that are running Westermo's operating system WeOS.Throughout the thesis, a Lynx-3510 has been used as the test platform. The switch is capable of handling both layer 2 and 3. The specific model used in this thesis work is a layer 3 switch. This enables it to not only act as a switch but is also capable of routing. The device has eight Ethernet ports that support 1 Gbit/s. The device also includes a pair of Small Form-factor Pluggable (SFP) connectors supporting the aforementioned speed. Equipped with Power over Ethernet (PoE), a console port, and a micro SD card reader. The console port can be connected to a computer through USB. This is used since the switch does not have a display and it makes it possible to configure and manage it. As for software, the device runs WeOS 5, which is a modern robust operating system that builds upon GNU/Linux. With this installed the device has support for virtualisation, web interface, IP routing, and VPN. The device operating voltage, depending on the model, is from 12 to 57 VDC. A more detailed list of specifications of the Lynx-3510 is listed in Table 3 and the device can be seen in Figure 11.

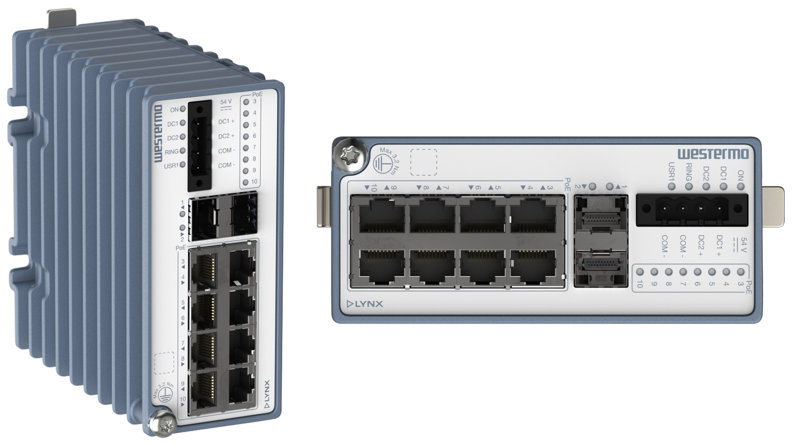


Figure 11: One of Westermo's devices, the Lynx-3510.

Family	Architecture	CPU	Hz (GHz)	Ram (MB)	Flash (MB)
Lynx-3510-F2G-P8G	ARMv8-A	NXP i.MX8 (nano)	1.4	512	128

Table 3: Hardware specifications for the Lynx-3510. The listed amount of resources is the total amount that is available on the device. Substantially less will be allocated to the container.

3. Related Work

Methods using machine learning are gaining popularity in protecting the expanding Internet. The papers presented in this section have utilised intrusion detection systems that can identify and flag abnormal activities.

Experiments in previous work show that high accuracy, when detecting network intrusions, can be achieved with RF. Ibrahim et al.[39] researched K-Nearest-Neighbour (KNN), RF, and Support Vector Machine (SVM) for detecting DoS, Probe, R2L and U2R attacks. Experiments using the NSL-KDD dataset show that RF has better accuracy for each attack tested and KNN resulted in produced results with better accuracy than SVM. Lu Zhou et al. [40], discussed that the current methods of classifying DDoS attacks have issues with privacy or inefficiency. The method used in the research removes redundant features and introduces a threshold for each feature. The algorithm applies an aggregated feature-based linear classifier to the features. Markovic et al. [41], used RF in a federated learning approach. A comparative study was conducted in order to find the best combination of hyperparameters, several DTs, splitting rule and ensemble method. The study involved multiple clients and one server. Each client independently trains on a subset of the dataset and sends the resulting model to the server that aggregates all the models. The intrusion detection datasets KDD, NSL-KDD, UNSW-NB15, and CIC-IDS-2017 were used. Research shows that combining independent RFs gives better accuracy than the average accuracy of each independent RF. This approach is used when the dataset is too large to be used in one node, but with it, there is a drawback of lower accuracy. Doshi et al. [42], conducted a study about detecting DDoS attacks originating from IoT devices. Packets grouped after sender and timespan were binary classified using different algorithms. All algorithms achieved good results and 99% accuracy, with RF being the most precise but KNN not far behind.

Leon et al. [43] conducted a study about ML for intrusion detection. The algorithms studied were of varied nature, both supervised learning and unsupervised learning.The validation was performed on four datasets containing multiple different types of attacks. Their paper concluded that RF was the best, based on accuracy, whereas KNN and SVM also gave promising results. The drawback with KNN was the execution time when classifying new cases.

Another important part in Intrusion Detection System (IDS) is how to convert network traffic into inputs for the ML model. Cirillo et al. [26], researched methods to classify packet flows as IoT traffic or non-IoT traffic. The system used information such as packet rate, size, and round trip time to combat encryption. The classification attributes focused on the number of acknowledged packets and bytes that flowed in each direction. Both simulated and real traffic was used as datasets [44], [45]. In the experiments, multiple different algorithms were evaluated with varying results with the J48 Classification tree peaking with over 99% accuracy. Naïve Bayes performed greatly classifying IoT but traditional traffic was often incorrectly classified as IoT.

Since IoT devices are becoming increasingly popular, much research has been done on how to protect these devices from cyber attacks. To protect IoT devices a Lightweight Intrusion Detection System (LIDS) is needed in these resource-constrained devices. Fenanir et al. [46] constructed a ML-based LIDS. They used three different datasets: KDD99, NSL-KDD and UNSW-NB15. Before sending the data to a ML algorithm they did feature selection. For the feature selection, they used a filter method, meaning that features are selected using a correlation matrix, Only the features where the correlation is greater than a threshold are selected. Their results show that DT and KNN performed better than other algorithms they used. KNN performed better but required more time to classify than the DT.

Leon et al. [47] studied another method to reduce features, namely, feature encoding. This approach builds upon an autoencoder, which is an Artificial Neural Network (ANN), designed for dimensional reduction. The autoencoder is trained so that it reduces the number of features that are calculated from the dataset before the data is sent to ML intrusion detection system. This system was validated using six different ML algorithms and different levels of reduction. The paper found that by reducing the number of features, time consumption decreased considerably, and without impacting accuracy too much.

RoyML et al. [48] proposed a new approach for a LIDS. The proposed model is based on an adaptive combination of boosting and stacking. The model consists of two levels. The first level is a simple classifier such as KNN, RF or Extreme Gradient Boosting (XGboost). The output of the first level is probabilities or predicted values. These values are then fed to the second-level classifier, which uses them as features in its training. On the CIC-IDS-2017 test dataset, the accuracy was 99.2%. While performing the classification, an Intel Core TM i5-9400F CPU 2.90GHz with 8GB RAM was used and the model consumed 272MB RAM (3.4%) and 1.5% to 2.9% utilisation of the CPU.

4. Method

In the following section, the research method is presented and the steps and methods taken in order to answer the research questions stated in Section 1.1. This study follows a comparative research methodology. In Figure 12, there is an overview of the steps taken to answer the research questions. In the beginning, this thesis focused on studying the state-of-the-art and trying to set up an LXC. During the study of the state-of-the-art, research papers focused on resource-constrained machine learning algorithms, anomaly detection, and network flows were examined.

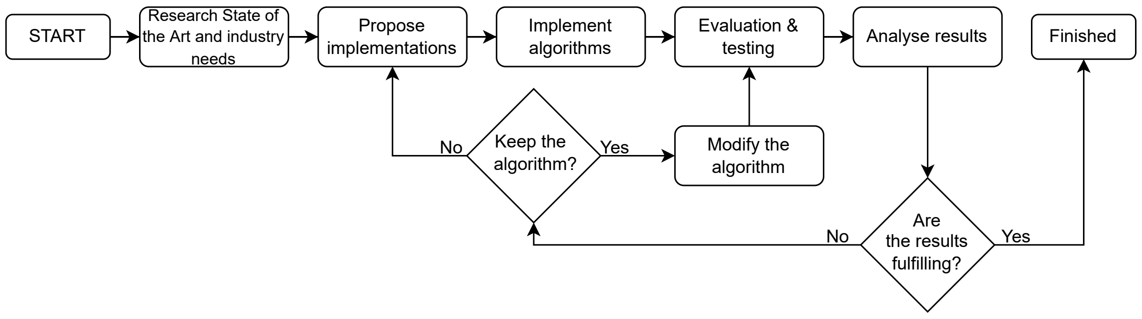


Figure 12: A flowchart visualising the workflow which is to be followed in order to answer the research questions.

The methodology flow is depicted in Figure 12, it is based on the research methodology presented in [49]. The methodology presents an approach that closely integrates the industry and its requirements into the steps taken. The first step is to identify the actual problem in the industry, which helps to improve the relevance and applicability of the research questions that are to be developed. A comparative methodology is appropriate because of the close relationship between this paper's research and Westermo's requirements.

A comparative research methodology emphasises the contrast in performance between various algorithms, rather than focusing solely on the outcome of a single algorithm. This also implies that the priority of the study is a quantitative analysis contrary to a qualitative one. Whereas the focus is to ensure that the results of the study are enough to be able to differentiate which algorithm applies to the kind of environment where it will be placed in.

From the basis of the previously mentioned research, multiple algorithms are studied and, depending on the outcome, implemented. Each implementation is then tested and evaluated. The testing should validate the logic of the models and ensure that their desired behaviour is consistent. Evaluation refers to calculating metrics, such as accuracy and precision on the validations set. Measurements of the allocated resources, such as execution time and memory, should also be evaluated during runtime. After evaluation, the algorithm will be modified and re-evaluated or, in case of a poor model, a new one will be created. This process continues until all the results are fulfilled or the time is out.

Throughout the work, a large portion of the results will be produced only by simulation. This is because the research that will be conducted builds upon the performance of ML. Due to that, the model does not need to run directly on hardware to validate the performance of the said model. Even if the models do not need to be evaluated on the switch, some models will be tested on the hardware in order to see how the models perform with live network traffic. The focus during this stage is not to validate the accuracy of the model, but instead, on the performance and functionality. This ensures that the program is working as intended, that the program function properly given the limited resources, and that the program is compatible with the system.

5. Implementation

The final implementation consists of numerous steps which are divided into two different programs. The first program, Flow Classification, is implemented in C and is developed to run inside the container on the network switch. The second program is developed in Python 3 and will train the classifiers on a desktop PC to produce a model. In Figure 13, a visual representation of this system is shown. Described below is a short summary and the following sections contain a more detailed description of each step.

1. Reading packets - The program supports reading packets from a PCAP file, that is, a file with pre-recorded packets, or live packets from a Network Interface Card (NIC).
2. Converting packets into flows - The program looks at the header of each packet that is read, to see if there exists a flow that the packet belongs to. If there is not, the packet becomes part of a new flow.
3. Calculating features - Here features are calculated for each flow. All features that may be calculated are listed in Table 4. This step also includes labelling, if exporting features.
4. Exporting the featured flows to a file - This step is optional. It is only active when the program is meant to generate a dataset. Therefore, the program does not export the flows if a model is loaded, and the classification is active. In this step, the flows with their features are exported to a file. The flows and their features are what make up a dataset.
5. Training a model - After the featured flows have been exported, they are used to train a model. Before training, a feature selection process and then a parameter tuning process is used. This step is programmed in Python 3 and is meant to run on a desktop PC.
6. Online dataset - When training, the program may use features from online datasets to train and generate a model, used as a substitute for the earlier steps.
7. Importing the model - The model that has been trained is imported back into the C program. The model includes a list of features that was decided during feature selection.
8. Classification - Classifying featured flows using the imported model on the device.
9. Act - The program responds by blinking with an LED when an attack is detected.

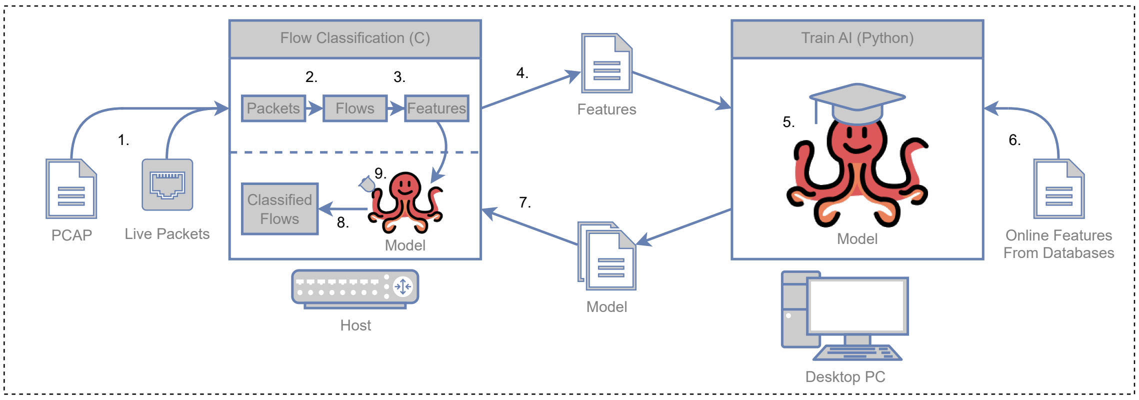


Figure 13: A visual representation of the implementation. Each number represents a step that is explained in the text.

For efficiency, the implementation developed during this thesis builds on multiple open-source libraries.

- Scikit Learn - A comprehensive Python 3 library, Scikit Learn[5], that is used extensively in the implementation when training the ML models. Scikit supports multiple algorithms, such

[5]https://scikit-learn.org/stable/

as DT, RF, and ANN providing easy access to train advanced models. Their implementation focus on maintainability and simplicity written in Python, building upon the NumPy library [50].

- Libpcap - An open-source library providing functions for reading packets and capturing network traffic from a wire. It is written in c, targeting c/c++ implementations supporting packet sniffing, modification, and transmission of packets. It is also implementing full support for reading and writing data to a PCAP file in offline mode [51].

5.1 Reading packets

Reading packets is accomplished using Libpcap. The implementation may either read from a file or directly from a NIC. In the former mode, a PCAP file should be provided containing pre-sampled packets. Each packet is then processed one at a time until the end of the file is reached. To read directly from a network interface and sample live packets, the program first lists all interfaces available and then asks the user from which one to read. From the chosen interface, packets are read until halted. It is possible for both configurations to specify the maximum amount of packets, which will terminate the reading when the said amount of packets has been reached.

Due to the program being placed in a container, additional configuration is needed to be set up for reading packets. Therefore a software bridge has been configured. The container inside the device possesses an external and an internal virtual port. The external port is then configured such that it is connected to the software bridge. Only ports that are connected to this bridge are in a position to forward their packets to the container. Therefore, each port that the model should listen to is connected to the bridge. Figure 14 is an example of such a configuration. In that case, all physical ports are connected, but other ports may exist that are not connected to the bridge, and are therefore not monitored by the model.

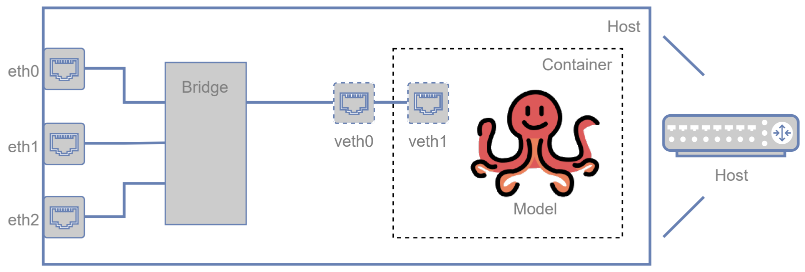


Figure 14: How the ports are configured on the Lynx-3510. The model is placed inside a container with a virtual network interface. Another virtual interface is then connected to the network bridge inside the device.

Inside the container, only a single network interface is visible, the virtual internal port, named veth1 in Figure 14. This port is bridged together with the external veth0 port and when configured correctly, all packets sent to the bridge, are forwarded to this interface.

5.2 Flows

The target device have limited computational resources. To reduce the load, packets are classified into flows. This is accomplished by looking at certain values in the packet header, namely: source IP address, destination IP address, destination port, and protocol as described in Section 2.2. If each value in a newly read packet is equal to an active flow, the packet is accounted to said flow. If this is not the case, and the new packet does not belong to an existing flow, the packet is placed in a new flow. When a new flow is created, the timestamp is saved. This sets the criteria for when a flow should end, and when no more packets will be appended to the flow. There are two reasons for ending a flow; flow idle timeout and flow active timeout. The former ends a flow when no more packets have been appended to the flow during a certain time span. On the other hand, the latter, ends a flow after a certain time has passed from when the flow was created. This is to avoid a flow spanning an exaggerated time span due to the consequent transmission of packets.

5.2.1 Labelling Strategies

Two labelling strategies were implemented. The first is called interval and the second is called attacks only.
Interval is an implementation which depends on a log file that is provided with the dataset. When the program reads from a PCAP file, it is possible to label the data using this log file. The log file contains the timestamp of when an attack starts and ends. The program then labels all flows that contain packets transmitted during an interval as an attack. A visualisation of the process is shown in Figure 15.

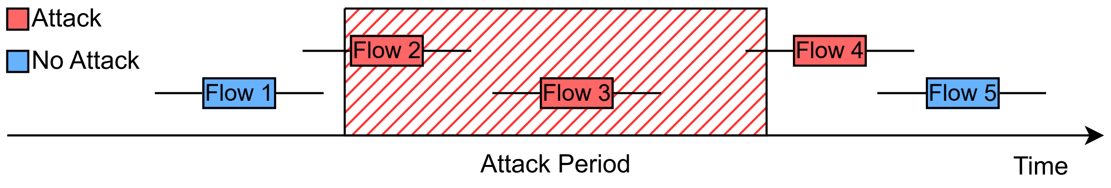


Figure 15: When labelling a flow, the program reads from a file containing the timestamps of the attacks. If a packet that is part of a flow, was transmitted during an attack, the flow is labelled as an attack. The flows in the figure marked as red are labelled as an attack, whereas the blue ones are not.

Attacks only is a more sparse labelling method. Instead of marking every packet that is transmitted during an attack as part of an attack, the strategy is to identify malicious packets and only mark those as part of an attack. The attacks are identified using a log file. If just a single packet in a flow is flagged, the flow is also flagged as an attack. This implementation depends on knowledge of the source of the attacks when labelling.

5.3 Datasets

Multiple datasets that are open source have been described in Section 2.6, but only the CIC-IDS-2017 dataset [36] and the Westermo dataset[6] were used. The CIC dataset is selected as it is the most recent one of those described in Section 2.6. Because the dataset is large, not all of the data was used, just the data recorded from one of the five days. Only the data from Friday is used which amounts to 8.3GB. This data only contains three types of attacks, Botnet, DDoS, and port scan. The CIC dataset is labelled and ready to be imported directly to train a model. But, because the features in the CIC dataset are already computed, there may be differences in how those features were calculated. If that is the case, it is hard to use the model trained in the classification program and still do a valid comparison between the models that was trained using the CIC dataset with the models trained on the Westermo dataset. Because of this, the raw data in PCAP format of the CIC dataset and the log files containing the timestamps of the attacks were used. The log files are used to label the data. Using the PCAP files made it possible to use the implemented features and create datasets based on those features. Before using the PCAP file, some preprocessing was applied. The preprocessing consists of converting packets into flows. From the CIC PCAP file, datasets were created using different flow duration and different labelling strategies. The flow duration was either 0.1s, 1s or 10s and the labelling strategy was either the interval labelling strategy or the attacks-only labelling strategy:

- CIC interval 10
- CIC interval 1
- CIC interval 0.1

[6]https://github.com/westermo/network-traffic-dataset

- CIC attacks only 10
- CIC attacks only 1
- CIC attacks only 0.1

From the PCAP files and log files that Westermo created, 18 datasets were derived using the packets recorded by the left device, the right device and the bottom device. For each of the three PCAP files, the flow duration is either 10s, 1s or 0.1s and uses either the interval labelling strategy or the attacks-only labelling strategy. To summarise, the following datasets were created:

- Bottom interval 10
- Bottom interval 1
- Bottom interval 0.1
- Bottom attacks only 10
- Bottom attacks only 1
- Bottom attacks only 0.1
- Right interval 10
- Right interval 1
- Right interval 0.1
- Right attacks only 10
- Right attacks only 1
- Right attacks only 0.1
- Left interval 10
- Left interval 1
- Left interval 0.1
- Left attacks only 10
- Left attacks only 1
- Left attacks only 0.1

5.4 Features

When a flow is complete, which happens after a certain time interval called flow duration mentioned in 5.2, the flow is exported or classified depending on the current configuration. The current implementation supports numerous features, building upon the IPFIX standard referenced in Section 2.2. All implemented features are listed in Table 4.

5.5 Exporting features

When exporting the features, the program produces a file with all the flows that have been read during the operation. The file begins with a number containing the number of features that have been computed and then a list of those features. Then each line contains the features for the corresponding flow in the order they have been completed. All features are listed in Table 4. Later, when training a model, features that are deemed unnecessary by the GA are removed.

Application		
ID	Feature	Description
1.	src port	Source port of the packets
2.	dst port	Destination port of the packets
3.	protocol identifier	protocol field in the IPv4 header
4.	ip class service	Value of the type of service field
Statistical		
ID	Feature	Description
5.	octet delta count	Total number of bytes
6.	packet delta count	Total number of packets
7.	flow duration	Delta time of first and last packet
8.	min ip total length	Smallest ip header + payload packet in the flow
9.	max ip total length	Largest ip header + payload packet in the flow
10.	ip header length	Length of the ip header
11.	flow byte rate	Amount of bytes per second
12.	flow packet rate	Amount of packets per second
13.	packet length min	The smallest packet in the flow
14.	packet length max	The largest packet in the flow
15.	packet length mean	The mean packet length
16.	packet length std	The standard deviation of the packet size
17.	flow IAT min	The minimum inter-arrival time of packets
18.	flow IAT max	The maximum inter-arrival time of packets
19.	flow IAT mean	The average inter-arrival time of packets
20.	flow IAT std	The standard deviation inter-arrival time of packets
21.	flow IAT total	The total inter-arrival time of packets
22.	mcast packet delta count	The amount of multicast packets
23.	mcast octet delta count	The total size of multicast packets
Flags		
ID	Feature	Description
24.	tcp control bits	Flags contained encoded bit fields
25.	igmp type	The Internet Group Management Protocol field
26.	icmp type code ipv4	Internet Control Message Protocol type and code field
27.	FIN flag count	Number of packets with "No more data from sender"
28.	SYN flag count	Number of packets with "Synchronise sequence numbers"
29.	RST flag count	Number of packets with "Reset the connection"
30.	PSH flag count	Number of packets with "Push Function"
31.	ACK flag count	Number of packets with "Acknowledgement"
32.	URG flag count	Number of packets with "Urgent"
33.	CWR flag count	Number of packets with "Congestion window reduced"
34.	ECE flag count	Number of packets with "ECN Echo"
35.	ipv4 Options	ipv4 option encoded in bit fields
36.	minimum TTL	The minimum time to live for the packets
37.	maximum TTL	The maximum time to live for the packets
38.	fragment offset	The data starting position for the packets
39.	fragment flags	Fragmentation properties of the packets
40.	ethernet type	The MAC client protocol of the payload

Table 4: List of features implemented. These are calculated for each flow and are exported to either the dataset for model training, or to classification.

5.6 Model Training

The training phase of the process is written in Python 3. This decision was made because of the larger availability of ML libraries. The implementation of the training is sustained using the library Scikit Learn described in Section 5.

Before training the model feature selection and parameter tuning is performed using a GA. Whereas before the GA is executed the dataset is randomly divided into a training set, validation set and test set. The training set composes 70% of the dataset, the validation set makes up 10% and the test set is 20%. Figure 16 shows a flowchart of the implemented GA. In this section, each step in the algorithm will be further described.

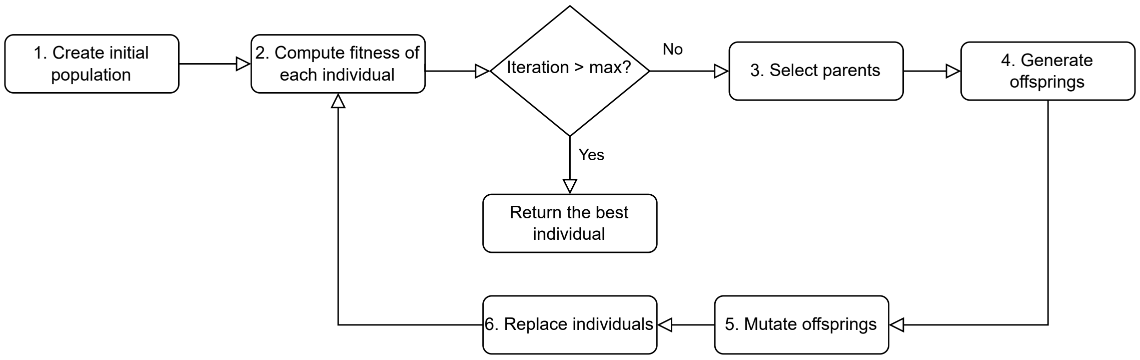


Figure 16: A flowchart of the implemented GA.

1. In the first step, an initial population of size 20 is created. The size of the population is specified as one of the input arguments. Each individual in the population is an array which can be used as input for the DT or RF classifier. The individuals are either binary arrays used for feature selection or an array with categorical values and real numbers used for parameter tuning. See Figure 17 for an example of the two individuals. The binary array used for feature selection has 40 elements since there are 40 features. Each element is either 1 or 0, where 1 means that the feature is to be used in the DT or RF and 0 means that the feature is not to be used. When an individual is initialised each element is assigned a 1 or a 0 with 50% probability.

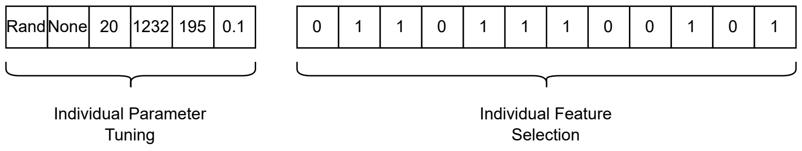


Figure 17: An example of the individuals in the implementation where feature selection and parameter tuning are carried out separately. In this example, there are 12 features in the dataset and the features 1, 2, 4, 5, 6, 9 and 11 are used in the DT/RF. In reality, the individual in feature selection has 40 elements and the individual in parameter tuning has 10 or 7 elements, 10 if the GA is used for tuning RF parameters and 7 if it is used for DT.

When using the GA for parameter tuning, the elements of the best individual are used as parameters for the DT or RF. Each element in that array is determined based on the range of the possible values on the hyperparameter. Also here the values are sampled randomly within the possible ranges. An individual used for parameter tuning of RF, X_{RF}, is the set of variables and D_{RF} is the domain of those variables. Any combination of values on the variables is allowed, so there are no constraints.

$$\begin{array}{ll}
X_{RF} = \{class\_weight, & D_{RF} = \{\{'balanced',' balanced\_subsample', None\}, \\
max\_features, & \{'sqrt',' log2', None\}, \\
n\_estimators, & \{x|x \in \mathbb{N}, 8 \leq x \leq 256\}, \\
max\_depth, & \{x|x \in \mathbb{N}, 10 \leq x \leq num\_samples\}, \\
min\_samples\_split, & \{x|x \in \mathbb{N}, 2 \leq x \leq 40\}, \\
min\_samples\_leaf, & \{x|x \in \mathbb{N}, 1 \leq x \leq 20\}, \\
max\_leaf\_nodes, & \{x|x \in \mathbb{N}, 2 \leq x \leq num\_samples\}, \\
max\_samples, & \{x|x \in \mathbb{N}, 10 \leq x \leq num\_samples\}, \\
min\_impurity\_decrease, & \{x|x \in \mathbb{R}, 0.0 \leq x \leq 0.01\}, \\
min\_weight\_fraction\_leaf\} & \{x|x \in \mathbb{R}, 0.0 \leq x \leq 0.01\}\}
\end{array}$$

The array for parameter tuning of DT has the following variable and domains:

$$\begin{array}{ll}
X_{DT} = \{splitter, & D_{DT} = \{\{'best',' random'\}, \\
class\_weight, & \{'balanced', None\}, \\
max\_features, & \{'sqrt',' log2', None\}, \\
max\_depth, & \{x|x \in \mathbb{N}, 1 \leq x \leq num\_samples\}, \\
min\_samples\_split, & \{x|x \in \mathbb{N}, 1 \leq x \leq 40\}, \\
min\_samples\_leaf, & \{x|x \in \mathbb{N}, 1 \leq x \leq 20\}, \\
max\_leaf\_nodes\} & \{x|x \in \mathbb{N}, 2 \leq x \leq num\_samples\}\}
\end{array}$$

2. Before calculating the fitness values each individual needs to be passed as input for the DT/RF classifier in order to determine the accuracy when using the individual's features and hyperparameters values. The fitness value of an individual is calculated by using that individual as input for the classifier. The classifier is either a random forest or a decision tree. If the genetic algorithm is to be used for feature selection, the accuracy of an individual is calculated by first removing the features not included in that individual. For example, if element i of the individual array is zero then column i in the training set and validation set is removed. This is repeated for all zero elements. Then, the classifier model is trained using the training dataset with removed columns and the fitness value of the individual becomes the accuracy that the model gets on the validation set with removed columns. Since bootstrapping is used for the random forest, each decision tree in the forest gets its own training dataset. These are taken randomly with replacement from the dataset the random forest uses as input. When doing feature selection, the dataset the random forest gets as input is the training set with removed columns. See the psuedo-code in Algorithm 3 for a description of this code implementation and the rest of the genetic algorithm. If the genetic algorithm is used for parameter tuning, the process is similar, were the difference is how the classifier is trained and which parameters it uses. In this case, the values of each of the elements in the individual is used as parameters for a classifier. Then the classifier is trained on the whole training set without removing columns, and the fitness value is the accuracy on the validation set.

   Then if this iteration is greater than the maximum number of iterations the best individual array will be returned, otherwise the next step is executed. The maximum number of iterations is 100.

3. The next step is to select parents. The parents are selected using Roulette selection. In Roulette selection, individuals with higher fitness values have a higher probability of being selected.

4. Two offspring are generated when doing a crossover. If the individual is a binary array or an array with categorical values, a single-point crossover is performed, otherwise BLX-α is used.

5. Mutation is done differently depending on the type of individual. If the individual is a binary array, a bit is flipped. If the element to be mutated is instead a categorical value, the value will be selected randomly from the domain of that element. All categorical values have the same probability of being selected. If the element is an integer or a float value, the mutation is carried out by adding a disturbance sampled from a normal density function. In this implementation, offspring1[i]=offspring1[i] + N($(x_{max}$-$x_{min})/2$+x_{min},$\frac{x_{max}-x_{min}}{6}$), where

x_{max} and x_{min} are the maximal and minimal values for element "i" of the individual. The mean is equal to $(x_{max}$-$x_{min})/2+x_{min}$ and the standard deviation is equal to $\frac{x_{max}-x_{min}}{6}$ to get a normal distribution like the one in Figure 18. In this Figure, the most probable value is the mean and min, max being the approximated boundaries.

The mutation rate is 1 divided by the length of the individual. Such that on average only one element mutates. In a later implementation, the mutation rate is 3 divided by the length of the individual.

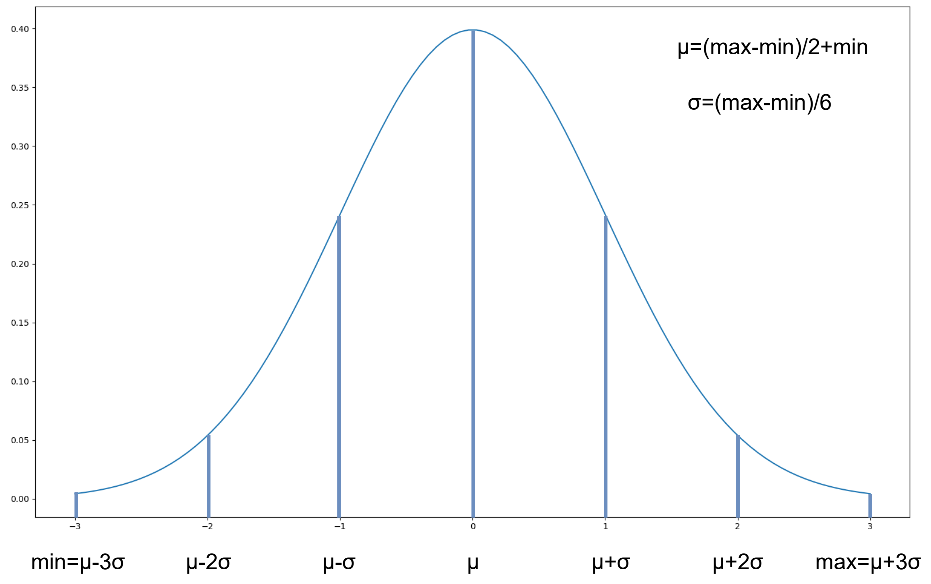


Figure 18: The normal distribution, N($(x_{max}$-$x_{min})/2+x_{min}, \frac{x_{max}-x_{min}}{6}$), the disturbance is sampled from when mutating a real number element. $\sigma = \frac{x_{max}-min}{6}$ since $\mu + 3\sigma - (\mu - 3\sigma) = 6\sigma \rightarrow max - min = 6\sigma \rightarrow \sigma = \frac{x_{max}-x_{min}}{6}$

6. The two offspring replace the two individuals with the lowest fitness values.

In this implementation, parameter tuning is first executed and then feature selection. During parameter tuning, all features are used each time the RF or DT is executed. Once the best individual from parameter tuning and the best individual from feature selection is obtained, a RF or DT classifier is trained with the values of the parameters as specified by the best individual from parameter tuning and the features selected as specified by the best individual from feature selection. Then this model containing one or several trees is exported as a text file. This text file is then imported to the classification program as described in Subsection 5.7.

5.7 Importing a Model

The model is read from the text file containing the trees and imported as binary search trees. The file must contain the number of features and a list of their names in the correct order. The program uses this information to match the information in the binary tree with its corresponding feature calculation function. This ensures that the model will work no matter which features that need to be calculated. It also ensures that the program does not calculate features that are not used. This saves computational power and speeds up the execution time. When importing a tree from the file, the program looks for either the end of the file or a marker that tells that a new tree is beginning. This ensures that each tree is handled as a separate tree. It is possible to import any amount and size of trees, as long as there is enough memory to load it into.

Algorithm 3 Genetic Algorithm for Feature Selection. When computing the fitness value for each individual, the columns of unwanted features are removed from the dataset. Then, the fitness for each individual is derived from the accuracy of model trained on the prepared dataset.

```
population = initialise_population()
while max_iteration not reached do
    for i = 0 i to size_population-1 do
        individual = population[i]
        selected_feat = find_all_zero_indexes(individual)
        new_train = remove_columns(train_set, selected_feat)
        new_val = remove_columns(val_set, selected_feat)
        train_classifier(new_train)
        y_predicted = predict(new_val)
        fitness_population[i] = accuracy(y_predicted, y_validation_set)
    end for
    parents = select_parents()
    offspring = crossover(parents)
    mutate(offspring)
    Remove the two individuals with the lowest fitness values
    Add the two offspring to the population
end while
return individual with highest fitness value
```

5.8 Classification

When a flow is completed, which happens when the flow duration has been reached, it is classified. When classifying, the program traverses the imported model using the features calculated for each flow. When multiple trees are loaded, it computes the decision of each tree and then exports the majority vote. Depending on the amount and size of the trees, the execution time will vary. Increasing the amount of trees, or the number of nodes in the trees will impact performance negatively. After the classification is done, the flow is deleted to free memory for new incoming flows.

5.9 Act

Currently, when the program is running on the device and detects an attack, no preventative action is taken. Instead, the device notifies the detection by using one of the LEDs next to the Ethernet ports. In Figure 19, a Lynx-3510 can be seen blinking because an attack was detected. For debugging purposes, when an attack is detected, the device uses the console port to transmit that a flow have been flagged as an attack. With this, information about the flow, such as the flow-defining attributes listed in Table 2, are also transmitted.

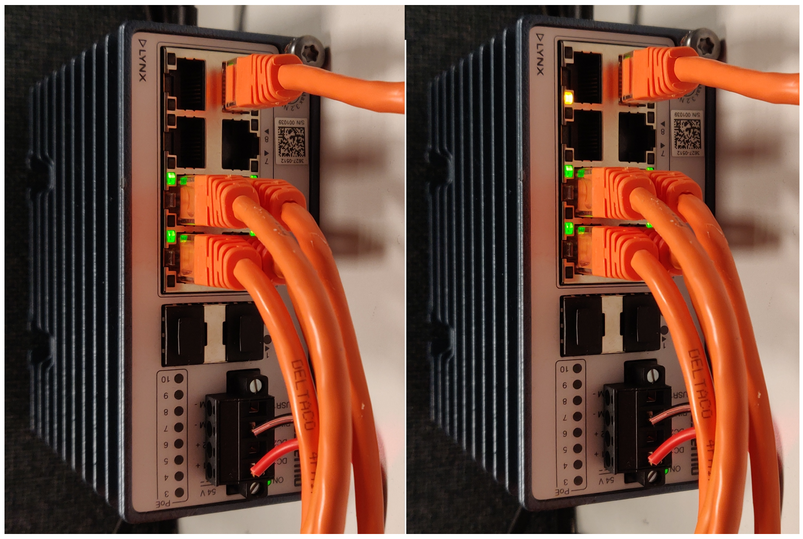


Figure 19: In the figure, the Lynx-3510 can be seen connected to the simulated factory. In the left picture is the Lynx-3510 during normal operation. But in the right picture, a LED is powered on an unused Ethernet port. This is to indicate that an attack has been detected.

5.10 Experimental Setup

To validate the model when running inside a container on a switch, a network with 5 Raspberry Pis, 2 Lynx-3510 and a PC was connected. The centre of the network is the Lynx-3510 loaded with the model. Each device is then connected to a corresponding Ethernet port on that Lynx-3510. See Figure 20 for the network topology that was used during the online validation.

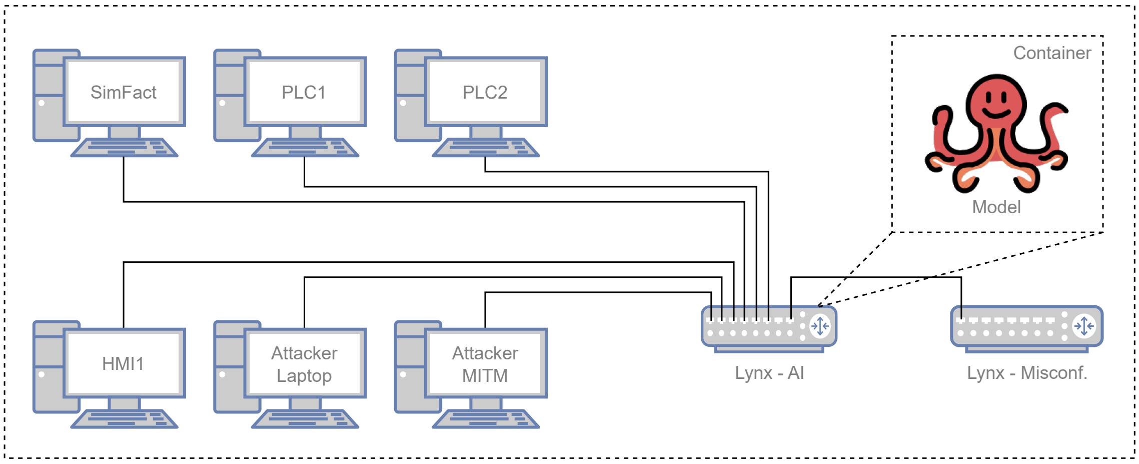


Figure 20: Topology of all devices included during the evaluation of the models. Included are five single-board computers, two Lynx-3510 and a laptop. One of the Lynx-3510 is hosting the model inside a container and all other devices are connected to this Lynx-3510 using Ethernet cables.

The attacks were launched from various devices during testing. Both port scan and the DoS attacks were launched from the PC marked as Attacker Laptop in Figure 20. The MITM were launched from the Raspberry Pi marked as Attacker MITM. The port scan was initialised using Nmap[7]. This searches the complete subnet, including all devices connected to the network, for ports that are open. The DoS attack was executed using Hping[8]. The virtual interface assigned to

[7]nmap -v 198.18.134.0/24
[8]sudo hping3 -S –flood -V -p 22 198.18.134.40

the container was configured to use the address "198.18.134.40". The MITM was undertaken by running a Python script from ICSSIM [52]. The misconfigurations were carried out by configuring the other Lynx-3510, marked in Figure 20 as Misconf. First, this Lynx-3510 was configured to use IP address 198.134.18.39 instead of 198.18.134.39 swapping the number in the middle of the IP address. The other misconfiguration is initialised by configuring the Lynx-3510 to use an address that already exists on the network. In this case, it was assigned to use the same address as one of the Raspberry Pis.

6. Results

Throughout the following chapter, all results are presented. The chapter is divided into three sections, offline feature selection, offline classification results, and online classification and performance. In the offline feature selection, results originating from the GA are presented. After which, in offline classification results, accuracies, and other metrics from model training are given. After these, the outcome from the online classification and performance of the models is presented. Here the models have been imported on the network switch, and connected to a simulated factory as described in Section 5.10.

6.1 Offline Feature Selection

The purpose of the results from training is to show how the fitness values or accuracies change with each generation. Only plots for some models are shown in the results. These are the training results from the model trained on data from the Westermo bottom with an interval labelling strategy and a flow duration of 10. The model was trained on the Westermo left dataset with an interval labelling strategy and flow duration of 10. The accuracy displayed in the graphs has been calculated on the validation set. A more extensive set of plots can be found in Appendix B.

6.1.1 Feature Occurrences

Before each model was trained, feature selection was applied using a GA, described in Section 5.6. Figure 21, shows how many times each feature was included in a model.

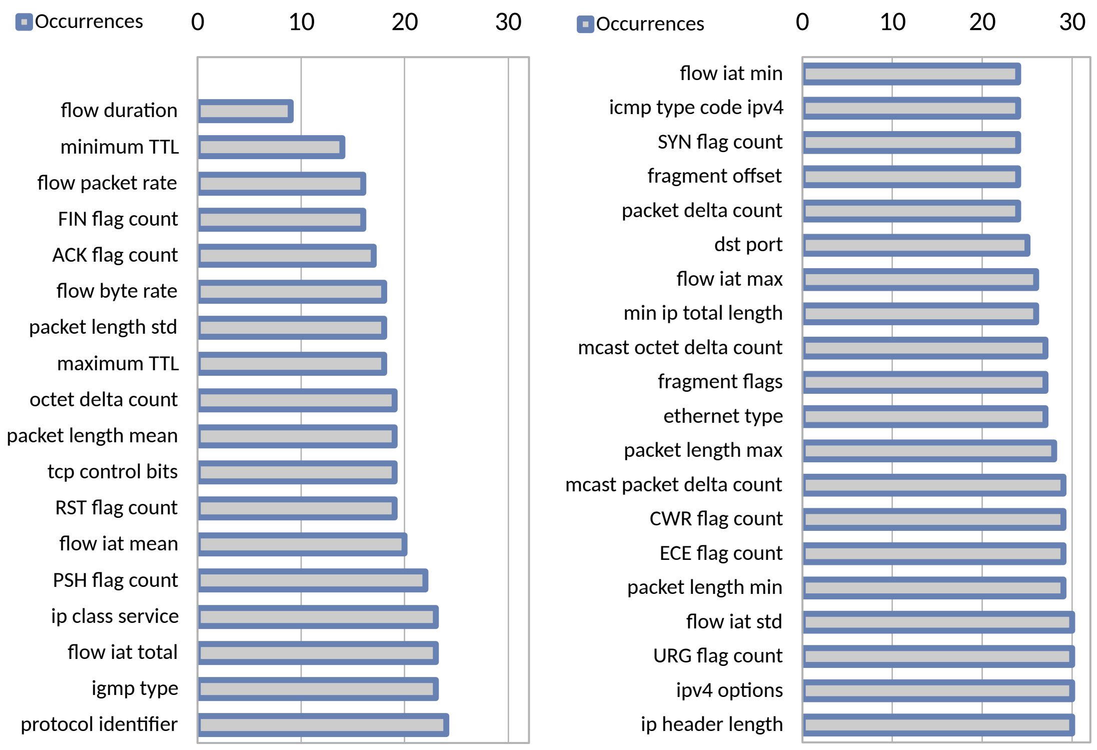


Figure 21: A collection of the features used from all the models evaluated above. For example, if a feature has 10 occurrences, it means that the feature has been used in 10 models.

6.1.2 Evaluation of Model bottom interval 10

In Figure 22, each column shows the result after running the GA for 100 generations and having a population size of 20 individuals. The dataset has been used for three different GAs. All the columns show the best and the mean value of each generation's accuracy and fitness value. The first two columns are the results from the GA used for feature selection. In the first column, the values of the weights are 0.85 and 0.15. These weights are used in the function that calculates the fitness values, see Subsection 5.6 for a reminder of how the weights work. In the second column, the values of the weights are 0.95 and 0.05. The last column shows the results from the GA used for parameter tuning. The GA used in Figure 22 had a mutation rate of 1/len(individual).

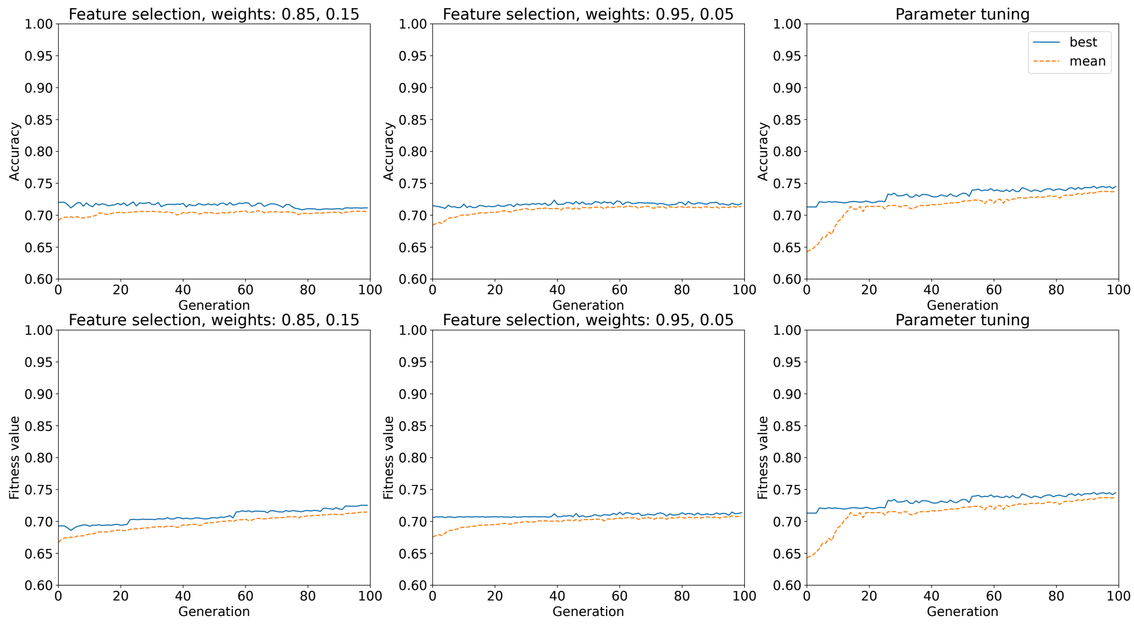


Figure 22: Evaluation of the GA for RF using the Bottom_Interval_10 dataset. The first row of subplots shows the best and the mean accuracy of the population in each generation. The first two plots in the row show the evaluation of the GA using different weights and the last plot in the row shows to the accuracy when using the GA for parameter tuning. The second row follows the same pattern but with fitness values instead of accuracies.

6.1.3 Evaluation model left interval 10

In the left subplots of Figure 23, 24, and 25, the GA is used for parameter tuning and in the subplots to the right the GA is used for feature selection. Here, the weights have values 1.0 and 0.0 when calculating the fitness values, so the fitness values equal the accuracies. Therefore, no plots of the fitness values are included.

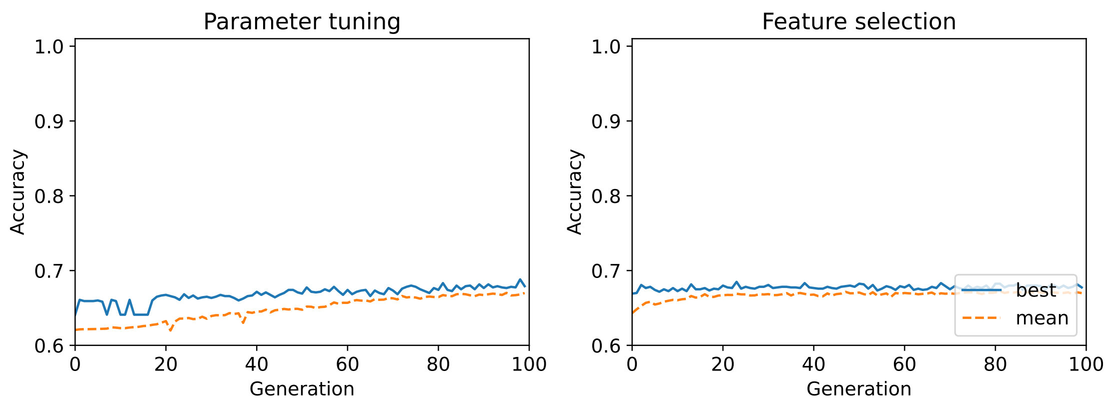


Figure 23: Evaluation of GA for feature selection and parameter tuning of a RF classifier. The dataset was the Left_Interval_10 and a mutation rate = 1/len(individual)

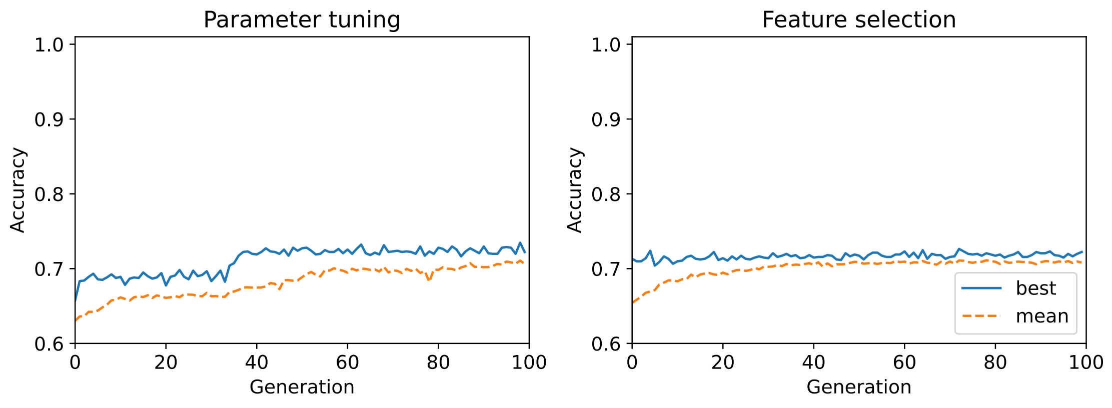


Figure 24: Evaluation of GA for feature selection and parameter tuning of a RF classifier. The dataset was the Left_Interval_10 and a mutation rate = 3/len(individual)

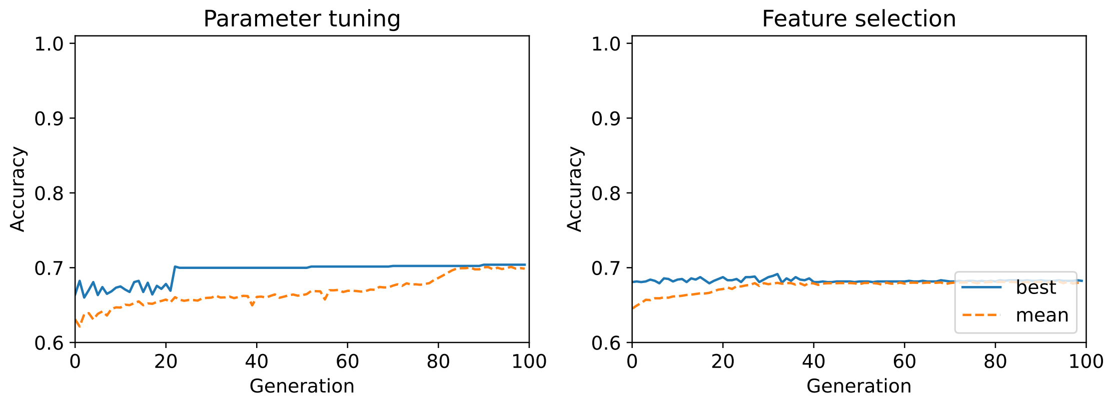


Figure 25: Evaluation of GA for feature selection and parameter tuning of a DT classifier. The dataset was the Left_Interval_10 and a mutation rate = 1/len(individual)

6.2 Offline Classification Results

This subsection presents various results gathered during training and testing of models on a PC. In Table 5, results from offline testing of different DT and RF models are presented. In Appendix A, a similar table presents additional information about the models, such as their size, the number of trees and the number of features selected. To present the results, the following values, originating from confusion matrices and derivations from the confusion matrices, are used:

- True Negative (TN), negative classified as negative
- False Positive (FP), negative classified as positive
- False Negative (FN), positive classified as negative
- True Positive (TP), positive classified as positive
- True Positive Rate (TPR), TPR = TP/(TP + FN)
- True Negative Rate (TNR), TNR = TN/(TN + FP)
- Balanced Accuracy (BA), BA = (TPR + TNR)/2
- Accuracy (Acc.), ACC = (TP + TN)/(TP + TN +FP + FN)

Even though the datasets in Table 5 are the same for different models, the division of the datasets into training set, validation set, and testing set is not the same. For example, a RF with mutation rate 1/len(individual) trained on Bottom_Interval_10 and a RF with mutation rate 3/len(individual) does not have the same training set, validation set and test set because of the random shuffle.

Evaluation Decision Tree, Mutation rate = 3/len(individual)										
Dataset	Labels	FD	TN	FP	FN	TP	TPR	TNR	BA	Acc.
Bottom	Interval	10	674	0	249	0	0.00	1.00	0.50	0.73
Right	Interval	10	306	5	132	11	0.08	0.98	0.53	0.70
Left	Interval	10	1312	239	522	340	0.39	0.85	0.62	0.68
CIC	Interval	10	23804	4157	4895	26400	0.84	0.85	0.85	0.85
Bottom	Interval	1	2811	0	742	0	0.00	1.00	0.50	0.79
Right	Interval	1	582	0	183	0	0.00	1.00	0.50	0.76
Left	Interval	1	2213	0	725	0	0.00	1.00	0.50	0.75
Bottom	Attacks	10	841	7	67	8	0.11	0.99	0.55	0.92
Right	Attacks	10	339	4	100	11	0.10	0.99	0.54	0.77
Left	Attacks	10	2200	3	198	12	0.06	1.00	0.53	0.92
Bottom	Attacks	1	3362	3	175	13	0.07	1.00	0.53	0.95
Right	Attacks	1	675	0	90	0	0.00	1.00	0.50	0.88
Left	Attacks	1	2728	3	196	11	0.05	1.00	0.53	0.93

Table 5: Evaluation of random forest and decision tree.

Evaluation Decision Tree, Mutation rate = 1/len(individual)

Dataset	Labels	FD	TN	FP	FN	TP	TPR	TNR	BA	Acc.
Bottom	Interval	10	679	0	244	0	0.00	1.00	0.50	0.74
Right	Interval	10	304	11	126	13	0.09	0.97	0.53	0.70
Left	Interval	10	1371	172	631	239	0.27	0.89	0.58	0.67
CIC	Interval	10	24011	3860	5155	26230	0.84	0.86	0.85	0.85
Bottom	Interval	1	2880	0	673	0	0.00	1.00	0.50	0.81
Right	Interval	1	605	0	160	0	0.00	1.00	0.50	0.79
Left	Interval	1	2210	0	728	6	0.01	1.00	0.50	0.75
Bottom	Attacks	10	855	4	64	0	0.00	1.00	0.50	0.93
Right	Attacks	10	353	8	92	1	0.01	0.98	0.49	0.78
Left	Attacks	10	2167	12	222	12	0.05	0.99	0.52	0.90
Bottom	Attacks	1	3355	4	185	9	0.05	1.00	0.52	0.95
Right	Attacks	1	650	7	102	6	0.06	0.99	0.52	0.86
Left	Attacks	1	2728	0	210	0	0.00	1.00	0.50	0.93

Evaluation Random Forest, Mutation rate = 3/len(individual)

Dataset	Labels	FD	TN	FP	FN	TP	TPR	TNR	BA	Acc.
Bottom	Interval	10	612	24	248	39	0.14	0.96	0.55	0.71
Right	Interval	10	317	3	132	2	0.01	0.99	0.50	0.70
Left	Interval	10	1251	303	457	402	0.47	0.81	0.64	0.69
CIC	Interval	10	24012	3867	5541	25836	0.82	0.86	0.84	0.84
Bottom	Interval	1	2819	0	734	0	0.00	1.00	0.50	0.79
Right	Interval	1	612	0	153	0	0.00	1.00	0.50	0.80
Left	Interval	1	2223	0	709	6	0.01	1.00	0.50	0.76
CIC	Interval	1	31782	4024	6009	40810	0.87	0.89	0.88	0.88
Bottom	Attacks	10	856	0	67	0	0.00	1.00	0.50	0.93
Right	Attacks	10	359	1	87	7	0.07	1.00	0.54	0.81
Left	Attacks	10	2168	19	182	44	0.19	0.99	0.59	0.92
CIC	Attack	10	58995	0	1	260	1.00	1.00	1.00	1.00
Bottom	Attacks	1	3338	0	215	0	0.00	1.00	0.50	0.94
Right	Attacks	1	657	7	95	6	0.06	0.99	0.52	0.87
Left	Attacks	1	2724	0	214	0	0.00	1.00	0.50	0.93
CIC	Attack	1	82255	0	4	366	0.99	1.00	0.99	1.00

Evaluation Random Forest, Mutation rate = 1/len(individual)

Dataset	Labels	FD	TN	FP	FN	TP	TPR	TNR	BA	Acc.
Bottom	Interval	10	671	0	250	2	0.01	1.00	0.50	0.73
Right	Interval	10	307	4	133	10	0.07	0.99	0.53	0.70
Left	Interval	10	1436	106	714	157	0.18	0.93	0.56	0.66
Bottom	Interval	1	2847	0	706	0	0.00	1.00	0.50	0.80
Right	Interval	1	588	6	160	11	0.06	0.99	0.53	0.78
Left	Interval	1	2159	31	697	51	0.07	0.99	0.53	0.75
Bottom	Attacks	10	858	1	60	4	0.06	1.00	0.53	0.93
Right	Attacks	10	365	0	89	0	0.00	1.00	0.50	0.80
Left	Attacks	10	2193	8	190	22	0.10	1.00	0.55	0.92
Bottom	Attacks	1	3348	4	194	7	0.03	1.00	0.52	0.94
Right	Attacks	1	673	0	92	0	0.00	1.00	0.50	0.88
Left	Attacks	1	2736	0	202	0	0.00	1.00	0.50	0.93

Table 5: Evaluation of random forest and decision tree. (Continued)

6.3 Online Classification and Performance

In this section, results from when running some ML models on the switch, as presented in 5.10. The result from this evaluation is summarised in Table 6. Only the models trained on the left dataset managed to detect attacks. They only managed to detect port scans and DoS attacks. In Figure 26, a diagram shows one of the attack sessions. Displayed are timings from when different attacks were launched and when the model classified flows as attacks. Models larger then 80 MB where not able to load into memory, and have therefor not been evaluated online.

Dataset	Labels	FD	Port S.	DoS	MITM	Misconf.	N trees	N feat.	Size
				Decision Tree					
Left	Interval	10	yes	yes	no	no	-	18	161kb
Bottom	Interval	10	no	no	no	no	-	15	162kb
Right	Interval	10	no	no	no	no	-	18	162kb
				Random Forest					
Left	Interval	10	yes	yes	no	no	79	17	1.20MB
Left	Interval	1	yes	yes	no	no	253	20	2.70MB
Left	Attacks	10	yes	yes	no	no	176	23	1.00MB
Left	Attacks	1	yes	no	no	no	119	14	28.5kB
Left	Attacks	0.1	yes	no	no	no	136	6	56.3kB
Right	Interval	10	no	no	no	no	67	19	13.0MB
Right	Interval	1	no	no	no	no	241	21	2.70MB
Right	Attacks	0.1	no	no	no	no	210	16	77.4MB
Bottom	Interval	10	no	no	no	no	111	19	46.0MB
Bottom	Interval	1	no	no	no	no	241	20	69.8MB

Table 6: The results from online verification on the device. The first column represents which datasets that were used for training. Attacks and intervals indicated the labelling strategy used, described in 5.2.1. Flow Duration (FD) refers to how long each flow is at max, in seconds. A yes means that at least one flow was successfully detected during an attack, while a no means that the model detected zero flows.

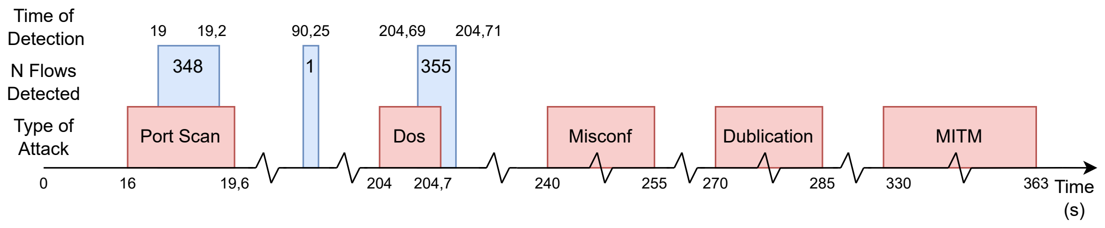


Figure 26: A sample of attacks launched towards the system topology, displayed in Figure 20. The model that was imported during the run was trained on Left_Interval_10. The timeline under the attacks, red rectangles, is used to display the timestamps for when an attack started and ended. The blue rectangles represent when the model classified attacks and the numbers inside are how many flows that were classified. The numbers on top of the blue squares are timestamps for the first and last detected flow.

7. Discussion

The previous section presented the results produced during the thesis. This section discusses said results and mentions decisions that were taken during the work, that could impact the results.

7.1 Genetic Algorithm

From the results, Section 6., it can be seen that the accuracy only increases a few percentage points and then gets stuck. The best individual in each generation does not change considerably. To lower the risk of getting stuck at a local minimum, the mutation rate and crossover can be increased. Increasing crossover can be done by creating more offspring or increasing the crossover points. Creating more than just two offspring that can replace the population would increase the variation of the individuals. Furthermore, the results show that the mean of the fitness values of the population increases more during the generations compared to the fitness value of the best individual.

Even though the fitness value did not depend on the number of features used, the GAs generated individuals using 50% of the features as the best individuals. This may depend on the fact that in feature selection an element of an individual is either 0 or 1 with a 50% probability and as mentioned before the individuals do not change that much. The results from the feature selections indicate that IP header length is the most important feature for detecting anomalies. The least important feature is flow duration, but the majority of the features seem to be of equal importance. In Figure 21 it can be seen that most of the features have a similar number of occurrences. Therefore, the results from the feature selections may also be random.

In Figure 22, weights were used to control how much the fitness values increase when the number of features decreased. One problem with this was that decreasing the number of features increased the fitness values too much. This can make the fitness value increase while decreasing the accuracy. This is bad for increasing accuracy. An example of this can be seen in Figure 22, where the first column of the plots, where the weights have values of 0.85 and 0.15.

7.2 Offline classification

Sometimes the trained models decide to classify everything as the majority class, the class representing no anomaly. This results in a high number of false negatives, which can be seen in Table 5. Many models result in a low TPR, less than 10%. These models can sometimes have high accuracy, but since they are not able to detect any or a few attacks their balanced accuracy is low. The models trained on the Westermo dataset that have a TPR greater than or equal to 10% are all the models trained on the Left_Interval_10 dataset, the RF models trained on the Left_Attacks_10 dataset and the RF model trained on Bottom_Interval_10 dataset using an increased mutation rate in the GA. The best-performing model is the RF model trained on the CIC dataset, with the attack labelling strategy and flow duration of 10. The model only found a single false negative, a balanced accuracy of 99.81% and an accuracy of 99.998%. That is a high accuracy compared to some Related works in Section 3.. But this thesis project only utilised data from one of the recorded days, which may have affected the results. The results in Table 5, suggest that the models trained on the data recorded by the left device in the Westermo dataset are better compared to the models trained on the data recorded by the other devices. This was expected since the bottom and the right device do not get attacked directly like the left device. The results also suggest that the model trained on datasets created using a flow duration of 10s performed better than models with a lower flow duration.

7.3 Online classification

The results from the live classification show that it is possible to classify packets from a network device using machine learning with some of the exported models. The only attacks that these models detected were DoS and port scan. No model was able to detect MITM or misconfigurations. This study compared random forest classifiers with decision tree classifiers and random forest managed to detect more attack flows and gave fewer false positives. If the exported models are less then 80 MB, they are able to run inside the container. Otherwise, the program do not have enough memory to function properly. If the random forest model does not have too many nodes in the forest they are preferred for detecting anomalies in a resource-constrained setting.

7.4 Research Questions

In this section, the authors reflect on the research questions, stated in Section 1.1.

- **Is it possible and if so, how well does random forest perform anomaly detection in a resource-constrained setting?**

  This question may be divided into two smaller questions. First of all, is it possible to perform anomaly detection in a resource-constrained setting? To be able to answer this question, the system described in Section 5. has been developed. This system includes a network device, that is limited in computational resources. During validation, multiple ML models were trained and later loaded onto the device. This device was placed in a network, described in Section 5.10, to validate the system. As documented in Section 6. results, online testing, both DoS and port scan have been detected. Misconfigurations and MITM were never detected by any model. Misconfigurations may be hard to detect because a misconfiguration is changing the IP address and the IP addresses were never included as a feature in the datasets.

  Secondly, how well does random forest perform anomaly detection in a resource-constrained setting? As described above, the RF were able to classify certain anomalies in the resource-constrained environment. Models larger then 80 MB could not load onto the device. There are multiple factors that may impact a model's size, but this suggests that smaller models are preferable. Thus it is important to implement further optimisations to the models.

- **Which actions are required in order to implement an anomaly detection system for network traffic on a switch with limited computational resources?** The main required actions in order to implement an anomaly detection system for network traffic on a switch with limited computational resources, are to find an appropriate dataset, preprocess the dataset if needed and then train the ML model. In Section 5. implementation, a suggestion for such a system has been developed. Another step that is important to save resources is converting packets into flows instead of reading each packet one by one. This is essential if there is a lot of traffic on the network.

  When implementing an anomaly detection system with limited resources, feature selection is another important step. When a RF model that had not done feature selection or parameter tuning was loaded into the switch, the program was only able to read 19 out of 100 trees, before it crashed because of insufficient memory available. Therefore, a necessary step in order to implement an anomaly detection system with limited resources is to tune the parameters or reduce the number of features. To reduce the number of features some feature selection needs to be performed. It is not necessary to tune the parameters and select the features with a GA, but that was the method used in this thesis project.

  Thus it has not been proven that all the developed actions are required. But, the system has been proven to be able to achieve classification for certain anomalies. But for the anomalies that were not detected, further research is needed for if additional steps are needed for detection or if the cause is elsewhere.

- **What information, features, in network traffic, help to improve the performance of the network traffic classification system?** Throughout the implementation, a total of 40 features were implemented as shown in Table 4. An essential tool used to narrow down these features was a GA. Figure 21 show the number of times each feature occurred in a model. Because the performance of each model is different, it is hard to tell, just from looking at Figure 21 to discern how useful a feature is. But it is clear which features are more frequently included, which points to these being more useful. It should also be noted that there exist more features than the total of 40 implemented, and more should be included to broaden the extent of the study.

8. Conclusions

The thesis work aimed to answer, if it is possible to detect network anomalies using ML from a network switch. The switches used in the study were modern, layer 3, devices provided by Westermo. The switches have support for containers, which act as a limited environment, to control and suppress the model's access to hardware. It is inside of a container where the ML model operates from.

Related works studied throughout the work pointed to the success of RF. This kind of algorithm is a lightweight model adapted for binary classifications. Success has been shown in anomaly detection and because of their basic nature, was chosen as a research subject for the thesis.

The system was developed into two separate programs with multiple subsystems. The two programs were a model trainer and a packet classifier.

The packet classifier, written in C, reads packets from the network interface of the device. Packets are then sorted into flows, and classified, using the basis attributes. For each flow, 40 different features are calculated, where these feature ranges from statistic, such as bytes per second, to attributes, like which protocol the packets follow. These flows and their corresponding features are what are later classified using the model that has been trained in the model trainer.

The model trainer, containing the GA and RF or DT was developed in Python 3. The GA was used for feature selection and for parameter tuning of the DT and RF. Features that are calculated for each flow, are later used as information for the ML algorithm to classify the packets. Because of this, fewer features are optimal to decrease processing time on the device. By implementing features selection using a GA, the number of features a trained model needed could be kept down.

To validate the work that was produced, the trained models were verified both offline and online. In the offline evaluation, confusion matrices and corresponding data have been analysed. During the online validation, a simulated factory has been connected together with two network switches. While the factory was running, a laptop launched different types of malicious attacks, including Port Scan and DoS. Another device injected a MITM. Also, non-malicious misconfigurations of IP addresses were executed.

Results from validation show that it is possible to run both DT and RF models directly on network hardware. The models were able to classify certain attacks. This should motivate further studies in increasing the accuracy, and optimise such a system.

9. Future Work

In the following section, are the author's own suggestions, according to their knowledge of the subject. The suggestions range from improving the current work to larger implementations that may deserve a project of their own.

The current implementation of the flow conversion and feature calculation processes described in Section 5.2 is implemented in such a way that a flow is identified from the four parameters shown in Table 2. This method does not take into account that packets flowing in the reverse direction could be part of the same flow. This means that two-way communication between two devices is organised as two distinct flows. If instead the packets were organised into a single flow such that a flow consisted of the packets in both directions, more features could be included. For example, features that compare the amount of data flowing in different directions. Because multiple attacks, such as DoS, consists of a large amount of information just flowing one way, it may be easier to distinguish such attacks if that information is included. The current implementation relies on the ML model finding this information itself. This could make it easier for the classification process.

The exported trees can be optimised by removing redundant branches and nodes if they lead to the same class. (Figure 27). This could decrease the size of the exported text file containing the model. The model text file needs to be in RAM all the time if it is going to be used on a Lynx-3510. If the text file is too big, the Lynx-3510 cannot read the whole file and the program in the container does not get the amount of memory it needs to function. Another solution is to store the text file on external memory like an SD card, but this would decrease the speed of the program.

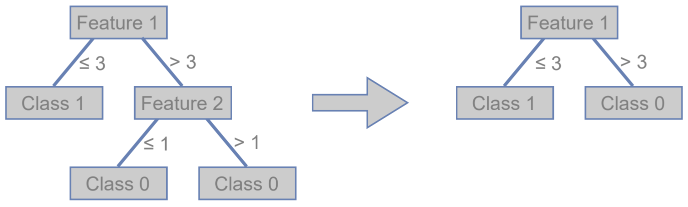


Figure 27: A proposed method of optimisation to the current implementation. The node "Feature 2" is removed since both leaves result in the same classification.

The GA can also be optimised. In the current implementation, the fitness values of the whole population are calculated in each generation even though it would be enough to just calculate the fitness values of the new offspring since the fitness of the other individuals has already been calculated. Calculating an individual's fitness value is a heavy operation because a classifier needs to be trained and evaluated on the validation set. This takes a long time if the training set is big. Since all the individual's fitness values, except the offspring's fitness values, are recalculated every generation, the accuracy of the best individual can change even though the best individual is the same. This is because the RF and DT contain randomness. This is why the accuracy can go down even though in each generation two new individuals replace the worst two in the population. Overall, the GA program needs to be improved. A bug was found in the mutation function when mutating real number elements. The disturbance which is added when mutating a real number element is sampled from a normal distribution with a different mean from the one intended. The mean is -(x_{max}-x_{min})/2 and not (x_{max}-x_{min})/2+x_{min} as it should be. This affects how the individuals are mutated in parameter tuning and may in turn affect the results.

An easy but time-consuming future work is to use more features. If there are more features the feature selection using GA will have a longer execution time. But the feature selection algorithm will have more features to choose from and this will increase the probability of selecting the features that increase the accuracy of the ML-models.

Creating a distributed intrusion detection system is another idea for future work. A distributed ML would be good in this case since no nodes in the network have all information required to detect all anomalies. If there are multiple computing nodes there are a lot of ideas that could be implemented. For example, having one ML for each type of attack, such that each model is specialised to find one kind of attack.

To expand the subject of operating an AI from a network device, it could be relevant to implement more ML algorithms. Especially techniques that are more distinct from the currently implemented models, such as ANN or KNN. This would not only be interesting as another method of detecting anomalies but also to reveal resource requirements for those models.

As with many projects, the implementation would benefit from optimisation, both for the Python 3 implementation and the C implementation. The latter should be especially important because the target system is a low-capacity device.

The current implementation has only been verified on the Lynx-3510. This may be enough to verify the product, but it would be interesting to determine the performance on different systems. Both systems with more resources, but also on systems with fewer resources to find lower limits. Also on the Lynx-3510, because some of the larger models were not able to load, because of limitations in system memory, it would be necessary to either optimise the load or use more powerful units.

As presented in Section 5., when an attack is detected, the model indicates this by blinking a LED on an Ethernet port. This is certainly enough to validate the model, but there exists a multitude of potential actions that could be taken. For example, the device should be able to take actions accordingly to the attack detected, like disabling a port or filtering the flagged packets.

10. Ethical and Societal Considerations

During this thesis, a large portion of the work revolves around analysing network data. There have also occurred data collection from the local network and simulated networks. Due to this, the work may have come in contact with personal data. Therefore it was important to take into consideration how the data are handled. But because of the nature of this thesis, the analyses of packets have not involved looking into the data that have been transmitted. Only the packet headers are being looked into. Also, no publication of any of the collected data has occurred.

The system that has been developed is designed to detect anomalies. Currently, no noteworthy action is taken when detection occurs, but that is a probable addition for future studies. Therefore, it is important to take into consideration what could happen if that is the case. There are many different ways to react to cyber intrusions. For example, notify the network manager, record suspect sessions, using a honeypot, discard suspicious network packets, kill the suspect process, or shut down the affected devices. If this authority is given to the program, other problems may occur if the program is not accurate. For example, if normal traffic is incorrectly classified as an attack, false positive. False positives can cause an unnecessary shutdown of a network or device. An attacker may also convince a reactive defence model that an entire network is under attack. This can result in a denial of service for innocent users. Or if this happens on critical infrastructure, such as a hospital, it may result in serious consequences. The same reasoning can be done for false negatives. If critical infrastructures are subject to cyber intrusion, this can cause serious harm as well.

References

[1] Y. Lu and L. D. Xu, ‘Internet of things (iot) cybersecurity research: A review of current research topics,’ *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2103–2115, 2019. DOI: 10.1109/JIOT.2018.2869847.

[2] R. Meier, A. Lavrenovs, K. Heinäaro, L. Gambazzi and V. Lenders, ‘Towards an ai-powered player in cyber defence exercises,’ in *2021 13th International Conference on Cyber Conflict (CyCon)*, 2021, pp. 309–326. DOI: 10.23919/CyCon51939.2021.9467801.

[3] E. Seker and H. H. Ozbenli, ‘The concept of cyber defence exercises (cdx): Planning, execution, evaluation,’ in *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2018, pp. 1–9. DOI: 10.1109/CyberSecPODS.2018.8560673.

[4] W. Tounsi and H. Rais, ‘A survey on technical threat intelligence in the age of sophisticated cyber attacks,’ *Computers & Security*, vol. 72, pp. 212–233, 1st Jan. 2018, ISSN: 0167-4048. DOI: 10.1016/j.cose.2017.09.001. (visited on 03/02/2023).

[5] T. Hwang, ‘Computational power and the social impact of artificial intelligence,’ 2018. DOI: 10.48550/ARXIV.1803.08971. (visited on 03/02/2023).

[6] G. Belani, ‘The use of artificial intelligence in cybersecurity: A review,’ Accessed: 2023-01-02. [Online]. Available: https://www.computer.org/publications/tech-news/trends/the-use-of-artificial-intelligence-in-cybersecurity.

[7] S. B. Wankhede, ‘Anomaly detection using machine learning techniques,’ in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, 2019, pp. 1–3. DOI: 10.1109/I2CT45611.2019.9033532.

[8] S. Hansman and R. Hunt, ‘A taxonomy of network and computer attacks,’ *Computers & Security*, vol. 24, no. 1, pp. 31–43, Feb. 2005, ISSN: 01674048. DOI: 10.1016/j.cose.2004.06.011. (visited on 19/01/2023).

[9] M. V. Pawar and J. Anuradha, ‘Network security and types of attacks in network,’ *Procedia Computer Science*, vol. 48, pp. 503–506, 2015, ISSN: 18770509. DOI: 10.1016/j.procs.2015.04.126. (visited on 19/01/2023).

[10] L. Cao, X. Jiang, Y. Zhao, S. Wang, D. You and X. Xu, ‘A survey of network attacks on cyber-physical systems,’ *IEEE Access*, vol. 8, pp. 44 219–44 227, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2977423. (visited on 19/01/2023).

[11] M. S. Elsayed, N.-A. Le-Khac, S. Dev and A. D. Jurcut, ‘DDoSNet: A deep-learning model for detecting network attacks,’ in *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, Cork, Ireland: IEEE, Aug. 2020, pp. 391–396, ISBN: 9781728173740. DOI: 10.1109/WoWMoM49955.2020.00072. (visited on 19/01/2023).

[12] S. T. Zargar, J. Joshi and D. Tipper, ‘A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks,’ *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013, ISSN: 1553-877X. DOI: 10.1109/SURV.2013.031413.00127.

[13] E. W. Felten, D. Balfanz, D. Dean and D. S. Wallach, ‘Web spoofing: An internet con game,’ 1997. [Online]. Available: https://www.cs.princeton.edu/research/techreps/TR-540-96 (visited on 03/02/2023).

[14] A. Bremler-Barr and H. Levy, ‘Spoofing prevention method,’ in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 1, Miami, FL, USA: IEEE, 2005, pp. 536–547, ISBN: 9780780389687. DOI: 10.1109/INFCOM.2005.1497921. (visited on 30/01/2023).

[15] J. Gadge and A. A. Patil, ‘Port scan detection,’ in *2008 16th IEEE International Conference on Networks*, ISSN: 2332-5798, Dec. 2008, pp. 1–6. DOI: 10.1109/ICON.2008.4772622.

[16] M. Dabbagh, A. J. Ghandour, K. Fawaz, W. E. Hajj and H. Hajj, ‘Slow port scanning detection,’ in *2011 7th International Conference on Information Assurance and Security (IAS)*, Dec. 2011, pp. 228–233. DOI: 10.1109/ISIAS.2011.6122824.

[17] G. Nath Nayak and S. Ghosh Samaddar, 'Different flavours of man-in-the-middle attack, consequences and feasible solutions,' in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 5, Jul. 2010, pp. 491–495. DOI: `10.1109/ICCSIT.2010.5563900`.

[18] R. Hofstede, P. Čeleda, B. Trammell *et al.*, 'Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX,' *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014, ISSN: 1553-877X. DOI: `10.1109/COMST.2014.2321898`.

[19] T. Zseby, B. Claise, J. Quittek and S. Zander, *Requirements for IP Flow Information Export (IPFIX)*, RFC 3917, Oct. 2004. DOI: `10.17487/RFC3917`.

[20] P. Aitken, B. Claise and B. Trammell, *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*, RFC 7011, Sep. 2013. DOI: `10.17487/RFC7011`.

[21] S. D'Antonio, T. Zseby, C. Henke and L. Peluso, *Flow Selection Techniques*, RFC 7014, Sep. 2013. DOI: `10.17487/RFC7014`.

[22] 'IP flow information export (IPFIX) entities.' (19th Aug. 2022), [Online]. Available: `http://www.iana.org/assignments/ipfix/ipfix.xhtml` (visited on 01/02/2023).

[23] M. Dash and H. Liu, 'Feature selection for classification,' *Intelligent Data Analysis*, vol. 1, no. 1, pp. 131–156, 1997, ISSN: 1088-467X. DOI: `https://doi.org/10.1016/S1088-467X(97)00008-5`.

[24] S. Picek, D. Jakobovic and M. Golub, 'On the recombination operator in the real-coded genetic algorithms,' in *2013 IEEE Congress on Evolutionary Computation*, ISSN: 1941-0026, Jun. 2013, pp. 3103–3110. DOI: `10.1109/CEC.2013.6557948`.

[25] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: from theory to algorithms.* NY, USA: Cambridge University Press, 2014, 397 pp., ISBN: 9781107057135.

[26] G. Cirillo, R. Passerone, A. Posenato and L. Rizzon, 'Statistical flow classification for the iot,' in *Applications in Electronics Pervading Industry, Environment and Society*, S. Saponara and A. De Gloria, Eds., Cham: Springer International Publishing, 2020, pp. 73–79. DOI: `10.1007/978-3-030-37277-4`.

[27] O. Maimon and L. Rokach, Eds., *Data Mining and Knowledge Discovery Handbook*, New York: Springer-Verlag, 2005, ISBN: 9780387244358. DOI: `10.1007/b107408`. (visited on 02/02/2023).

[28] G. Biau and E. Scornet, 'A random forest guided tour,' *TEST*, vol. 25, no. 2, pp. 197–227, 1st Jun. 2016, ISSN: 1863-8260. DOI: `10.1007/s11749-016-0481-7`. (visited on 20/01/2023).

[29] E. B. K. Thomas G Dietterich, 'Machine learning bias, statistical bias, and statistical variance of decision tree algorithms,' 1995. [Online]. Available: `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=893b204890394d1bf4f3332b4b902bfdb30a9a13` (visited on 26/01/2023).

[30] L. Breiman, 'Random forests,' *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: `10.1023/A:1010933404324`. (visited on 20/01/2023).

[31] L. Breiman, 'Bagging predictors,' *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug. 1996, ISSN: 0885-6125, 1573-0565. DOI: `10.1007/BF00058655`. (visited on 20/01/2023).

[32] T. M. Oshiro, P. S. Perez and J. A. Baranauskas, 'How many trees in a random forest?' In *Machine Learning and Data Mining in Pattern Recognition*, P. Perner, Ed., red. by D. Hutchison, T. Kanade, J. Kittler *et al.*, vol. 7376, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 154–168, ISBN: 9783642315367. DOI: `10.1007/978-3-642-31537-4_13`. (visited on 20/01/2023).

[33] 'OpenML.' (7th Oct. 2014), [Online]. Available: `https://www.openml.org/search?type=data&sort=runs&id=1113&status=active` (visited on 30/01/2023).

[34] 'NSL-KDD | datasets | research | canadian institute for cybersecurity | UNB.' (2023), [Online]. Available: `https://www.unb.ca/cic/datasets/nsl.html` (visited on 30/01/2023).

[35] N. Moustafa and J. Slay, 'UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),' in *2015 Military Communications and Information Systems Conference (MilCIS)*, Nov. 2015, pp. 1–6. DOI: `10.1109/MilCIS.2015.7348942`.

[36] 'IDS 2017 | datasets | research | canadian institute for cybersecurity | UNB.' (2017), [Online]. Available: `https://www.unb.ca/cic/datasets/ids-2017.html` (visited on 24/02/2023).

[37] P. E. Strandberg, D. Söderman, A. Dehlaghi-Ghadim *et al.* 'The Westermo network traffic data set.' (2023), [Online]. Available: `https://github.com/westermo/network-traffic-dataset` (visited on 11/04/2023).

[38] S. Graber. 'What's lxc?' Accessed: 2023-04-03. (24th Aug. 2022), [Online]. Available: `https://linuxcontainers.org/lxc/introduction/`.

[39] Z. K. Ibrahim and M. Y. Thanon, 'Performance comparison of intrusion detection system using three different machine learning algorithms,' in *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, 2021, pp. 1116–1124. DOI: `10.1109/ICICT50816.2021.9358775`.

[40] L. Zhou, Y. Zhu, T. Zong and Y. Xiang, 'A feature selection-based method for DDoS attack flow classification,' *Future Generation Computer Systems*, vol. 132, pp. 67–79, Jul. 2022, ISSN: 0167739X. DOI: `10.1016/j.future.2022.02.006`. (visited on 19/01/2023).

[41] T. Marković, M. L. Ortiz, D. Buffoni and S. Punnekkat, 'Random forest based on federated learning for intrusion detection,' in *Artificial Intelligence Applications and Innovations*, I. Maglogiannis, L. Iliadis, J. Macintyre and P. Cortez, Eds., ser. IFIP Advances in Information and Communication Technology, Cham: Springer International Publishing, 2022, pp. 132–144, ISBN: 9783031083334. DOI: `10.1007/978-3-031-08333-4_11`.

[42] R. Doshi, N. Apthorpe and N. Feamster, 'Machine learning DDoS detection for consumer internet of things devices,' in *2018 IEEE Security and Privacy Workshops (SPW)*, May 2018, pp. 29–35. DOI: `10.1109/SPW.2018.00013`.

[43] M. Leon, T. Markovic and S. Punnekkat, 'Comparative evaluation of machine learning algorithms for network intrusion detection and attack classification,' in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 01–08. DOI: `10.1109/IJCNN55064.2022.9892293`.

[44] R. Fontugne, P. Borgnat, P. Abry and K. Fukuda, 'Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking,' 2010.

[45] A. Sivanathan, H. H. Gharakheili, F. Loi *et al.*, 'Classifying IoT devices in smart environments using network traffic characteristics,' *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019, ISSN: 1558-0660. DOI: `10.1109/TMC.2018.2866249`.

[46] S. Fenanir, F. Semchedine and A. Baadache, 'A machine learning-based lightweight intrusion detection system for the internet of things,' *Revue d'Intelligence Artificielle*, vol. 33, no. 3, pp. 203–211, 10th Oct. 2019, ISSN: 0992499X, 19585748. DOI: `10.18280/ria.330306`. [Online]. Available: `http://www.iieta.org/journals/ria/paper/10.18280/ria.330306` (visited on 26/02/2023).

[47] M. Leon, T. Markovic and S. Punnekkat, 'Feature encoding with autoencoder and differential evolution for network intrusion detection using machine learning,' in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '22, Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 2152–2159, ISBN: 9781450392686. DOI: `10.1145/3520304.3534009`.

[48] S. Roy, J. Li, B.-J. Choi and Y. Bai, 'A lightweight supervised intrusion detection mechanism for IoT networks,' *Future Generation Computer Systems*, vol. 127, pp. 276–285, Feb. 2022, ISSN: 0167739X. DOI: `10.1016/j.future.2021.09.027`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/S0167739X21003733` (visited on 27/02/2023).

[49] T. Gorschek, P. Garre, S. Larsson and C. Wohlin, 'A model for technology transfer in practice,' *IEEE Software*, vol. 23, no. 6, pp. 88–95, Nov. 2006, ISSN: 0740-7459, 1937-4194. DOI: `10.1109/MS.2006.147`. (visited on 26/01/2023).

[50] L. Buitinck, G. Louppe, M. Blondel *et al.*, 'API design for machine learning software: Experiences from the scikit-learn project,' in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[51] 'Programming with pcap.' (2002), [Online]. Available: `https://www.tcpdump.org/pcap.html` (visited on 21/03/2023).

[52] A. Dehlaghi. 'ICSSIM.' (2023), [Online]. Available: `https://github.com/AlirezaDehlaghi/ICSSIM` (visited on 30/04/2023).

A Additional Offline Results

Evaluation Random Forest, Mutation rate = 1/len(individual)					
Dataset	Labels	Flow Duration	Number of trees	Number of features	Size
Bottom	Interval	10	31	21	28.4kB
Bottom	Interval	1	98	17	2.2kB
Right	Interval	10	26	15	32.7kB
Right	Interval	1	39	16	318.3kB
Left	Interval	10	89	21	1.1MB
Left	Interval	1	79	21	19.7MB
Bottom	Attacks	10	21	21	35.7kB
Bottom	Attacks	1	28	21	1.9MB
Right	Attacks	10	39	17	8.7kB
Right	Attacks	1	94	21	8.9kB
Left	Attacks	10	65	18	3.5MB
Left	Attacks	1	39	18	2.3B
Evaluation Random Forest, Mutation rate = 3/len(individual)					
Dataset	Labels	Flow Duration	Number of trees	Number of features	Size
Bottom	Interval	10	8	22	110kB
Bottom	Interval	1	128	15	5.00kB
Right	Interval	10	149	16	610kB
Right	Interval	1	256	20	1.20MB
Left	Interval	10	200	20	190MB
Left	Interval	1	34	14	1.70MB
Bottom	Attacks	10	8	16	78.2kB
Bottom	Attacks	1	11	19	0.51kB
Right	Attacks	10	68	20	175kB
Right	Attacks	1	256	24	4.10MB
Left	Attacks	10	103	16	20.8MB
Left	Attacks	1	15	13	495MB
CIC	Interval	10	9	16	152MB
CIC	Interval	1	8	16	208.3MB
CIC	Attack	10	8	25	71.5kB
CIC	Attack	1	8	25	92.9kB
Evaluation Decision Tree, Mutation rate = 1/len(individual)					
Dataset	Labels	Flow Duration	Number of trees	Number of features	Size
Bottom	Interval	10	-	16	352bytes
Bottom	Interval	1	-	16	371bytes
Right	Interval	10	-	13	9.4kB
Right	Interval	1	-	18	1.0kB
Left	Interval	10	-	18	47.8kB
Left	Interval	1	-	16	327bytes
Bottom	Attacks	10	-	17	36.1kB
Bottom	Attacks	1	-	14	53.5kB
Right	Attacks	10	-	19	4.3kB
Right	Attacks	1	-	18	24.3kB
Left	Attacks	10	-	19	36.3kB
Left	Attacks	1	-	14	724bytes
CIC	Interval	10	-	20	47.1MB

Table 7: Additional data from the Evaluation of random forest and decision tree models.

Evaluation Decision Tree, Mutation rate = 3/len(individual)					
Dataset	Labels	Flow Duration	Number of trees	Number of features	Size
Bottom	Interval	10	-	20	404bytes
Bottom	Interval	1	-	13	304bytes
Right	Interval	10	-	16	1.5kB
Right	Interval	1	-	15	315bytes
Left	Interval	10	-	13	224.5kB
Left	Interval	1	-	9	224bytes
Bottom	Attacks	10	-	18	30.9kB
Bottom	Attacks	1	-	13	52.1kB
Right	Attacks	10	-	16	10.2kB
Right	Attacks	1	-	17	2.0kB
Left	Attacks	10	-	14	163.0kB
Left	Attacks	1	-	20	354.1kB
CIC	Interval	10	-	16	47.7MB

Table 7: Additional data from the Evaluation of random forest and decision tree models. (Continued)

B Results on validation and test set

2.1 Evaluation models Westermo Right

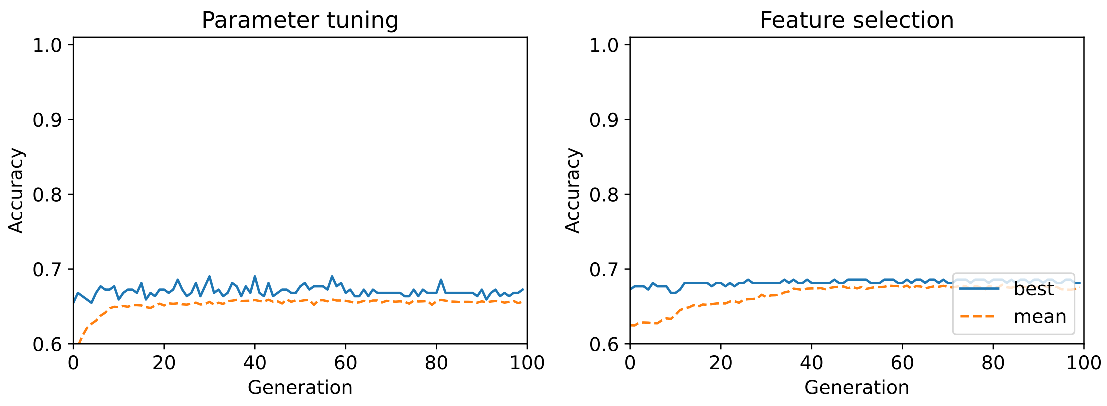


Figure 28: RF right interval 10, mutation rate = 1/len(individual)

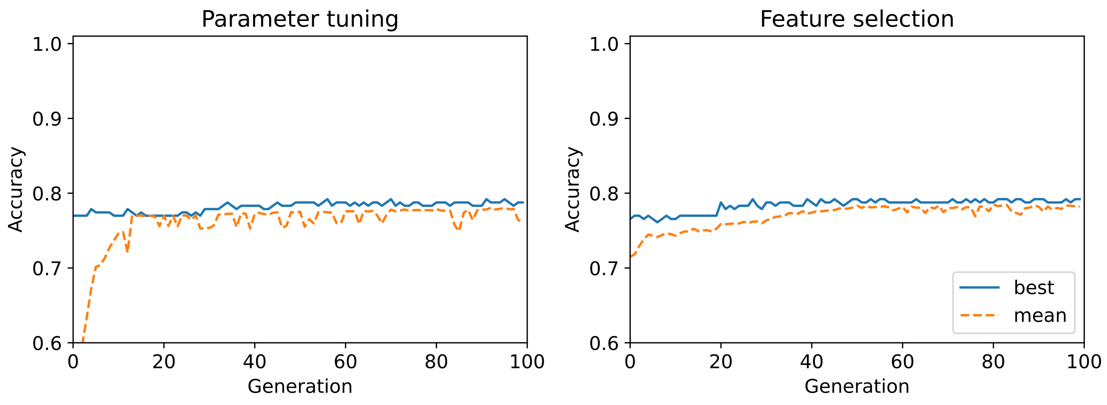


Figure 29: RF right interval 10, increased mutation, mutation rate = 3/len(individuals)

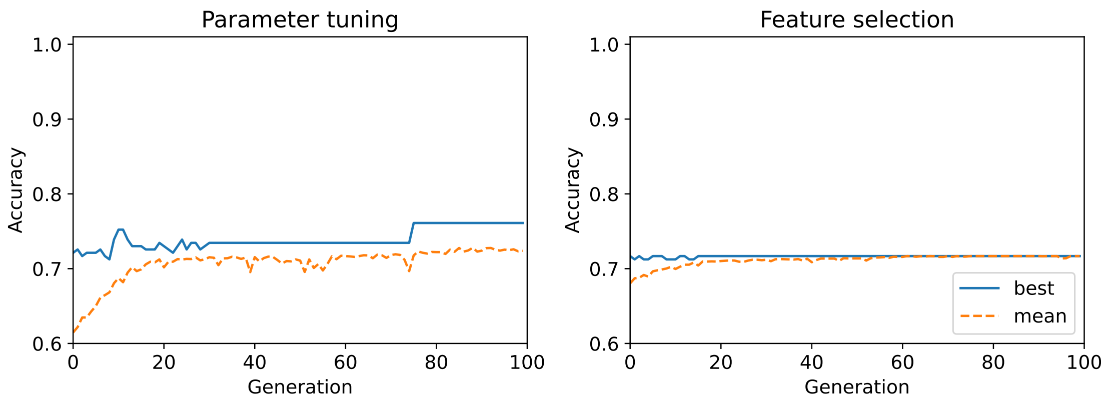


Figure 30: DT right interval 10, mutation rate = 1/len(individuals)

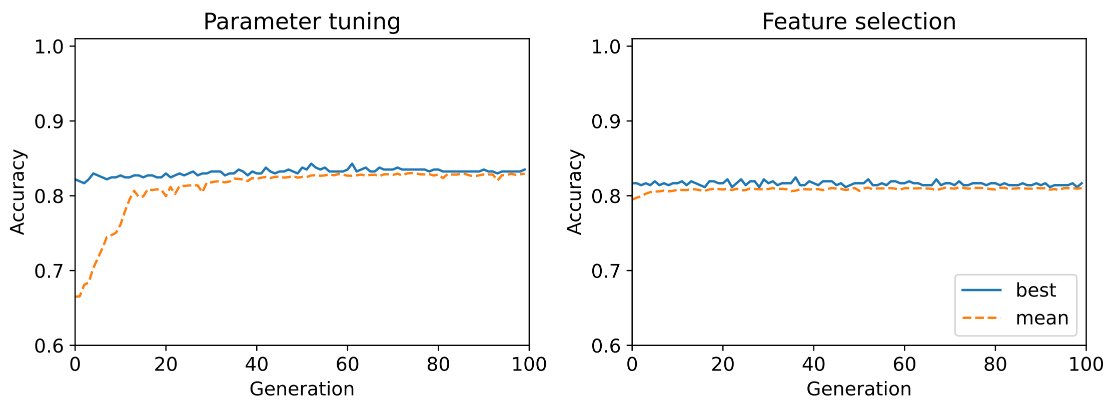


Figure 31: RF right interval 1, mutation rate = 1/len(individuals)

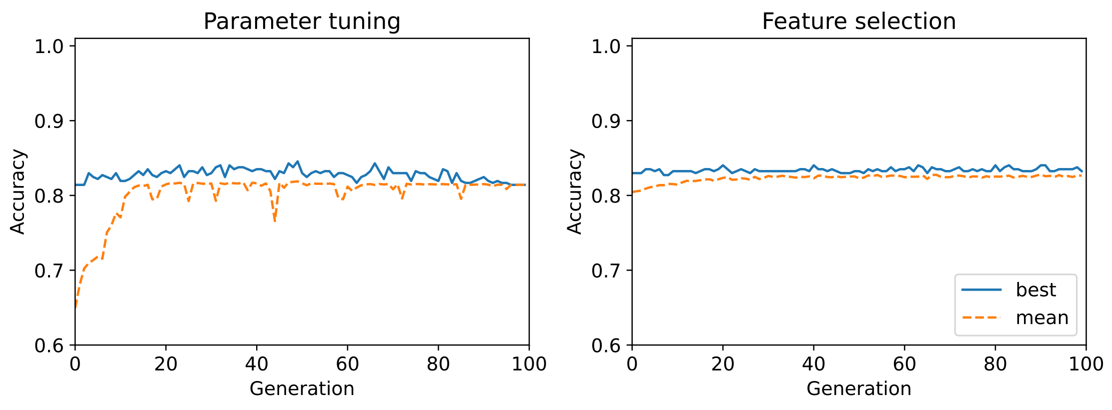


Figure 32: RF right interval 1, increased mutation, mutation rate = 3/len(individuals)

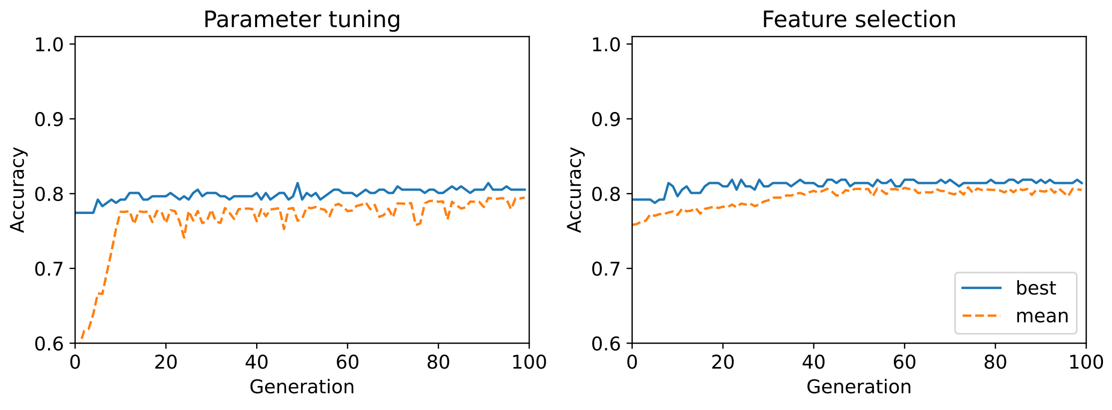


Figure 33: RF right attack 10, mutation rate = 3/len(individuals)

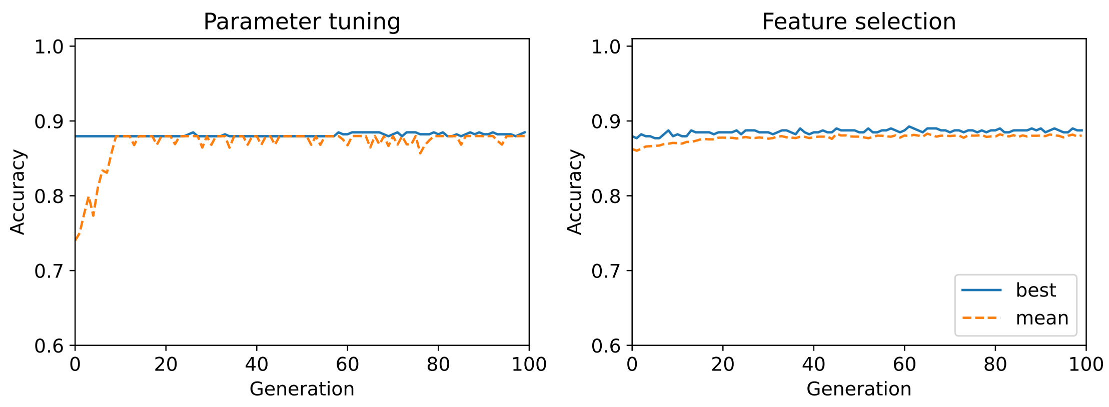


Figure 34: RF right only attacks 1, mutation rate = 3/len(individuals)

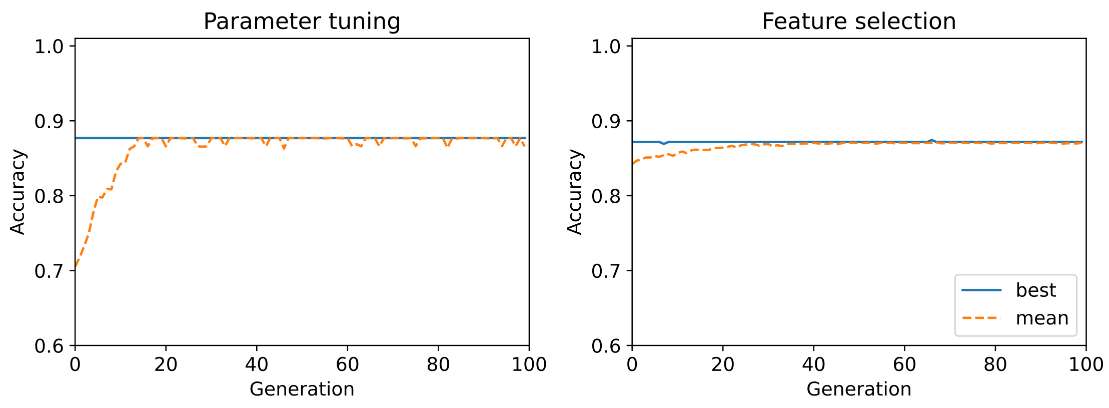


Figure 35: RF right attack 1

2.2 Evaluation models Westermo Bottom

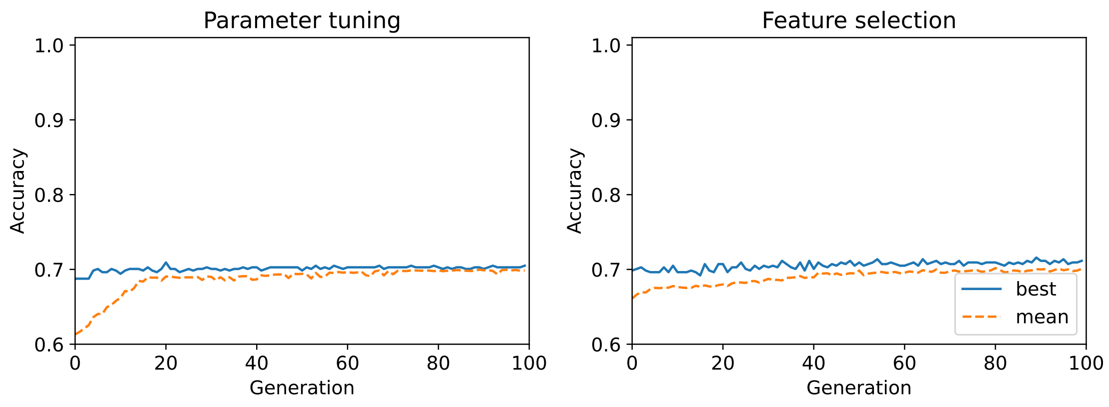


Figure 36: RF bottom interval 10, mutation rate = 1/len(individual)

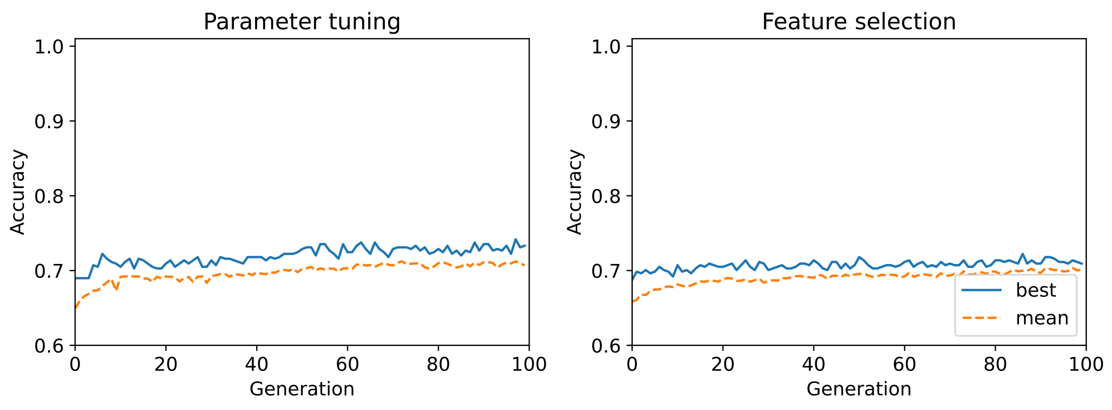


Figure 37: RF bottom interval 10, increased mutation, mutation rate = 3/len(individual)

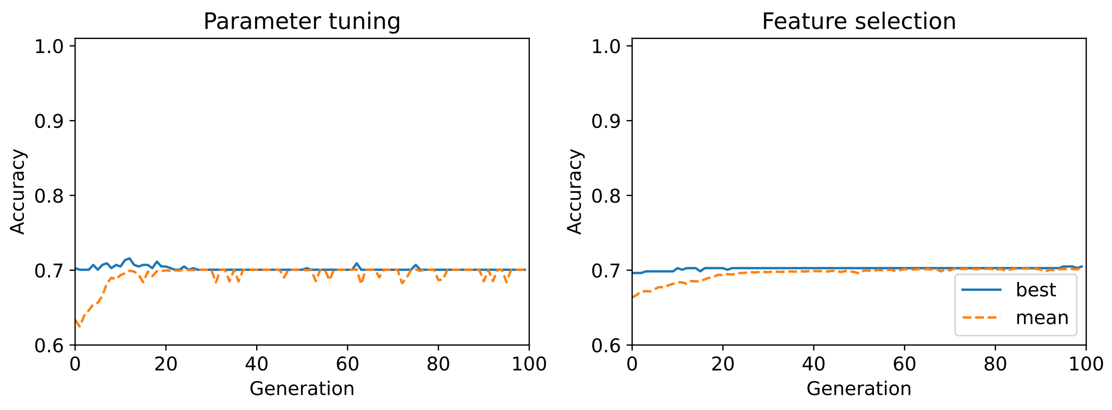


Figure 38: DT bottom interval 10, mutation rate = 1/len(individual)

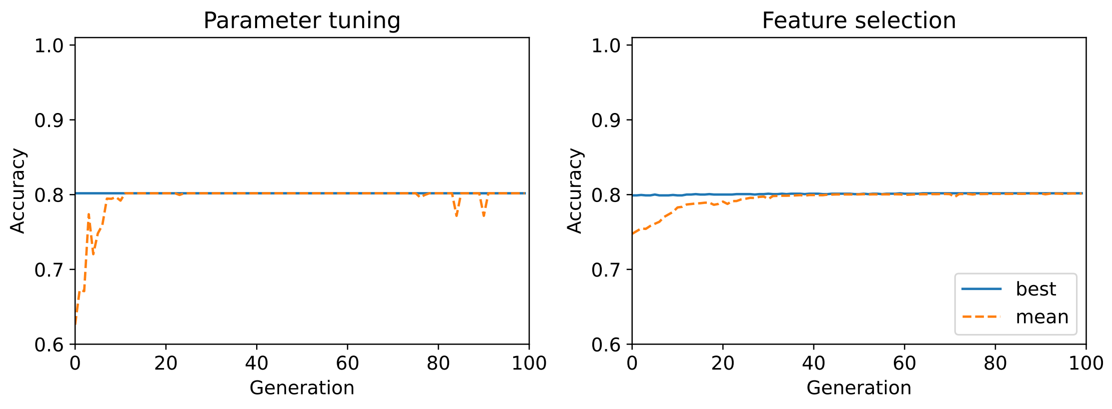


Figure 39: RF bottom interval 1

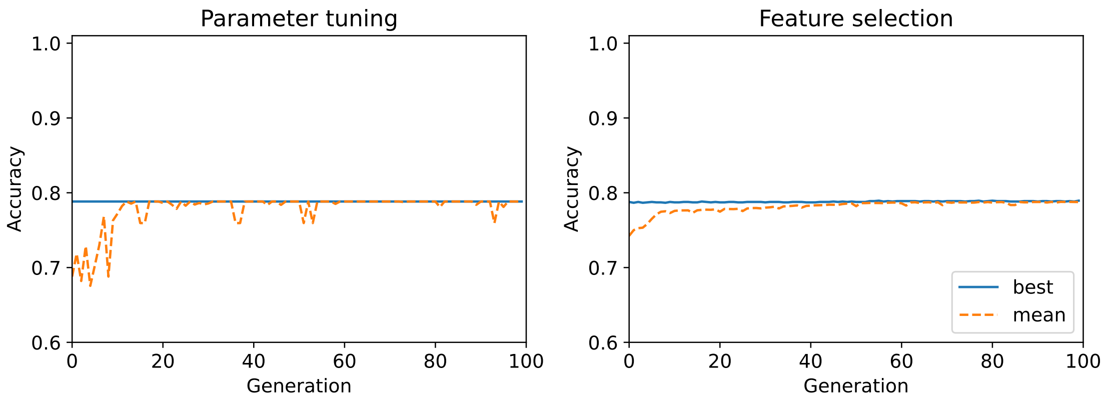


Figure 40: RF bottom interval 1, mutation rate = 3/len(individual)

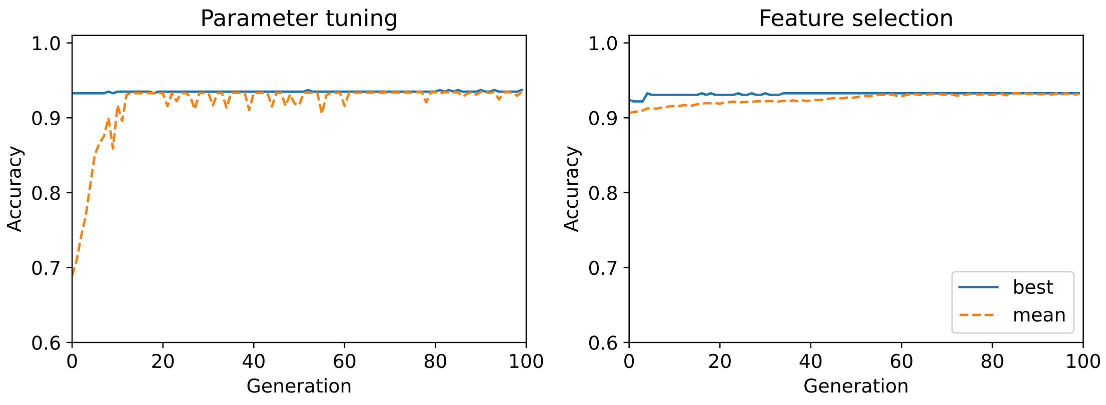


Figure 41: RF bottom only attacks 10, mutation rate = 3/len(individual)

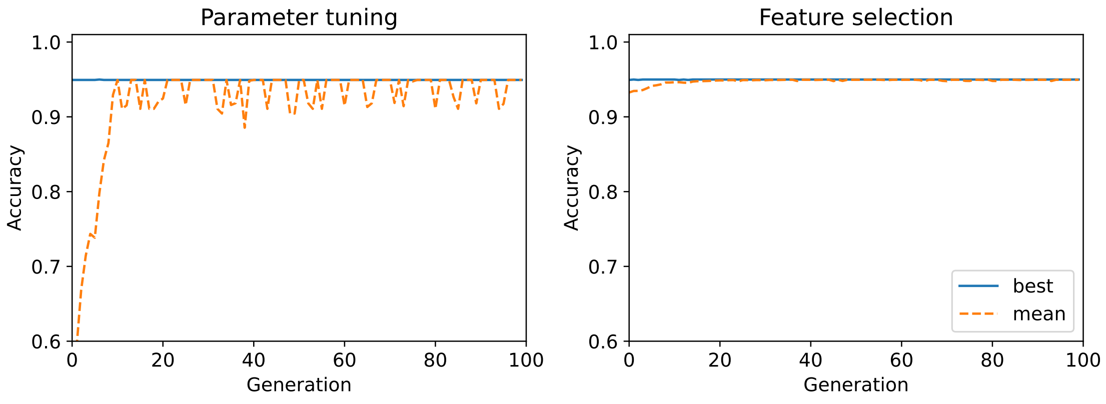


Figure 42: RF bottom only attacks 1, mutation rate = 3/len(individual)

2.3 Evaluation models Westermo Left

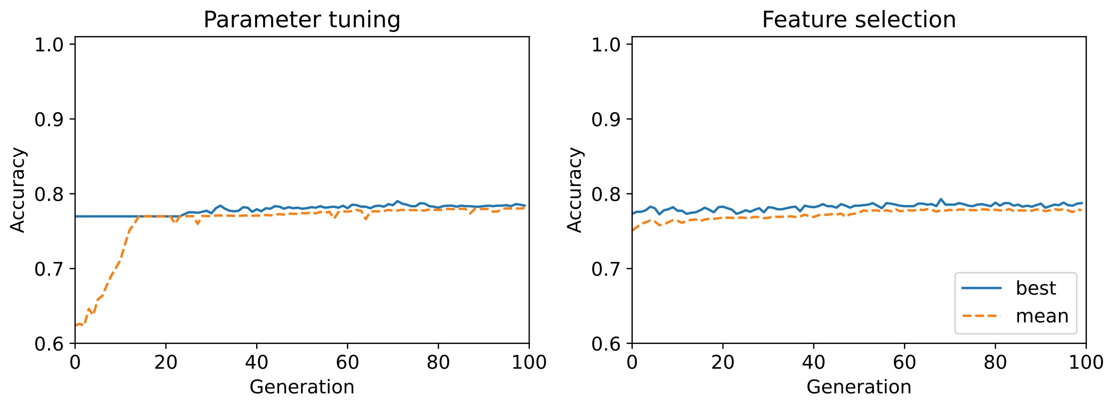


Figure 43: RF left interval 1

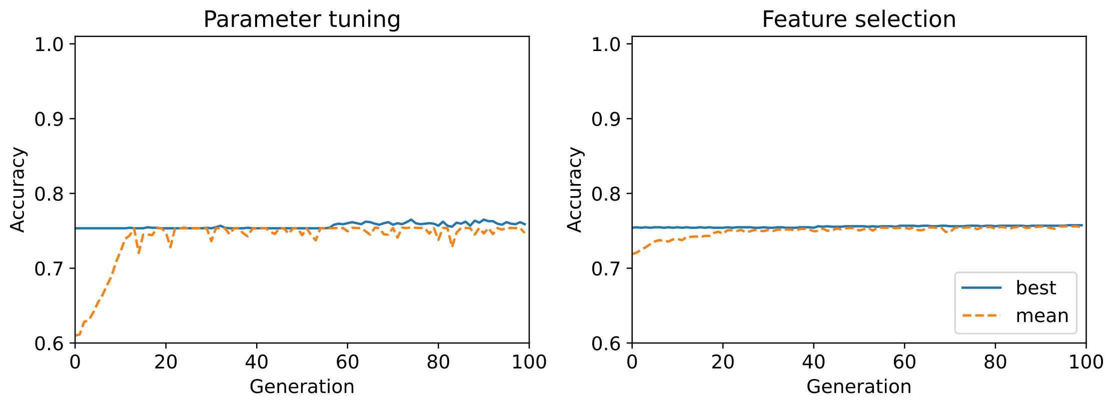


Figure 44: RF left interval 1, mutation rate = 3/len(individual)

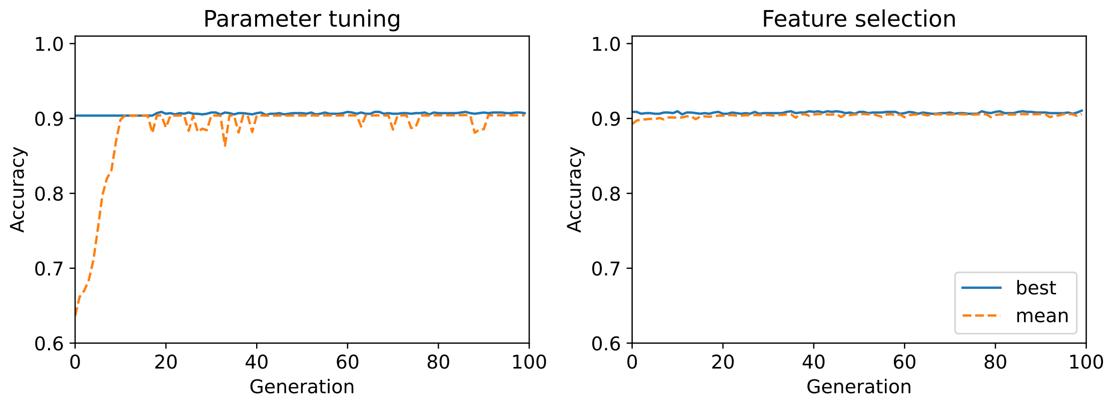


Figure 45: RF left attacks only 10

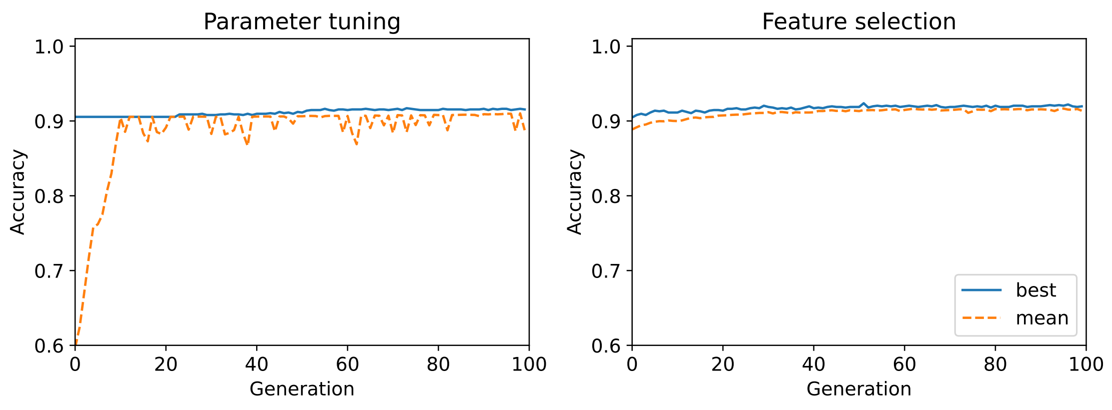


Figure 46: RF left attack only 10, mutation rate = 1/len(individual)

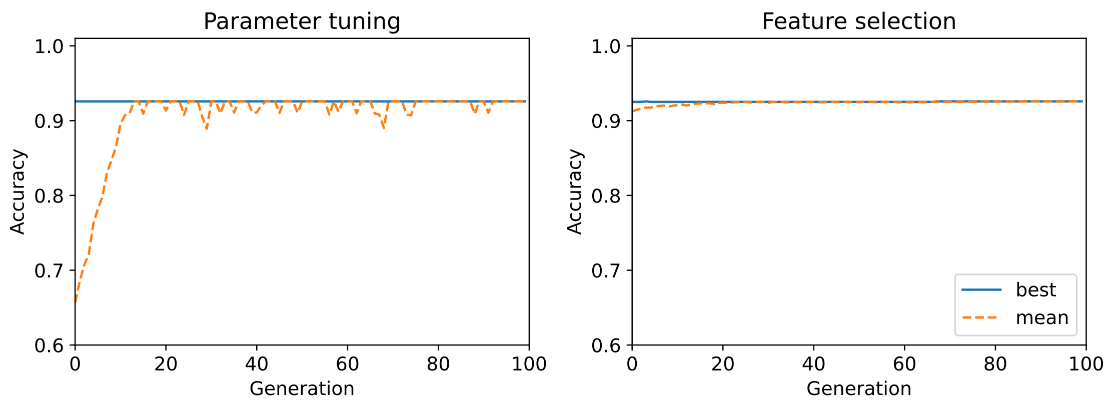


Figure 47: RF left attacks only 1

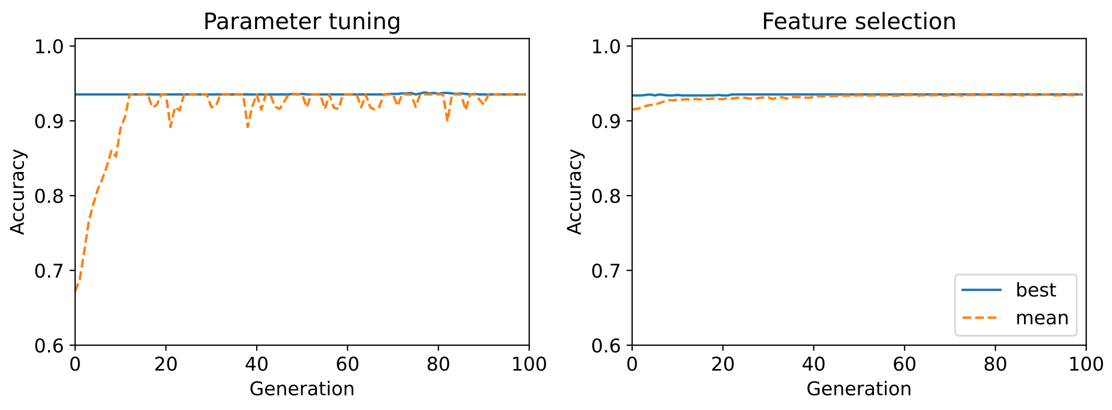


Figure 48: RF left attack only 1, mutation rate = 3/len(individual)

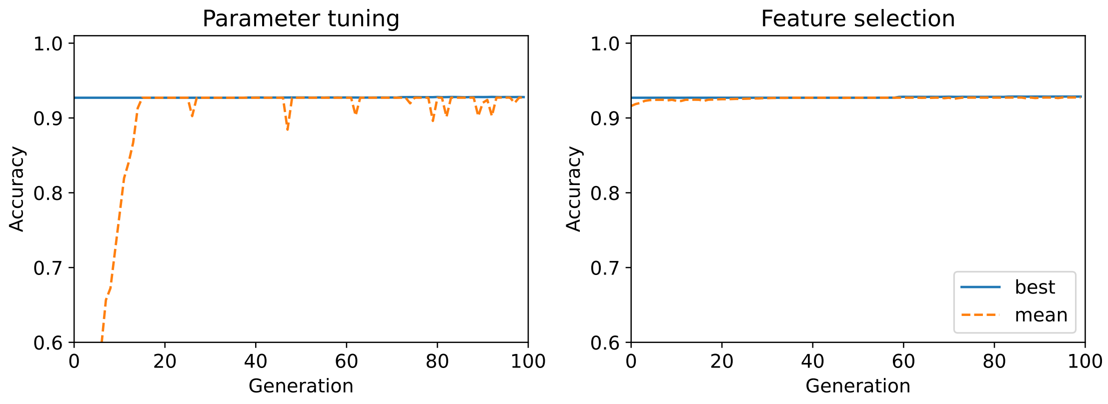


Figure 49: RF left attacks only 0.1

2.4 Evaluation models CIC

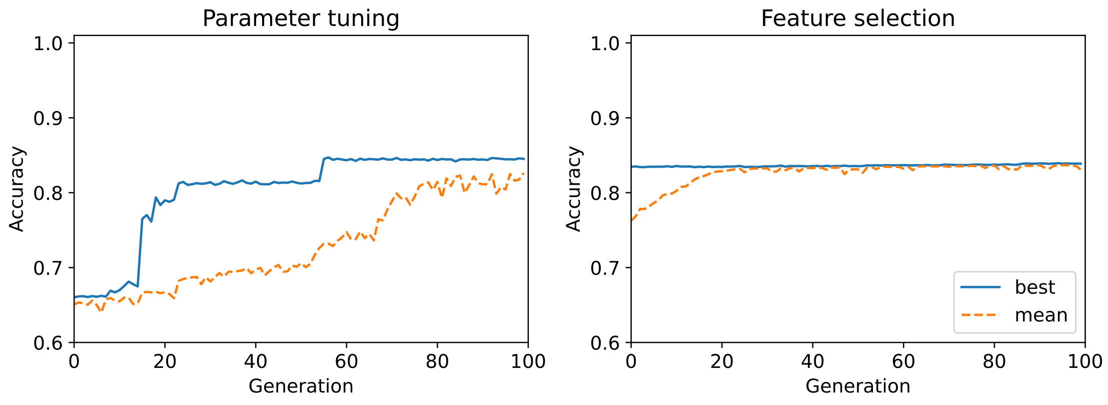


Figure 50: RF cic interval 10, mutation rate = 3/len(individual)

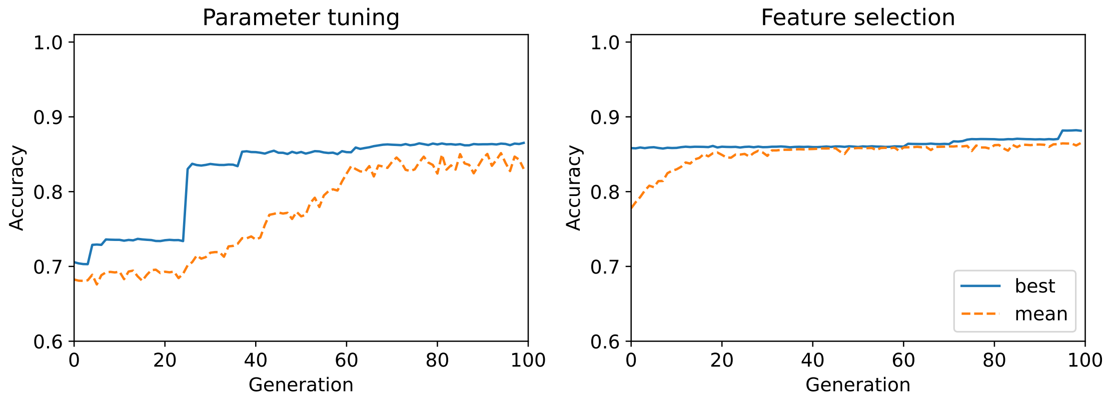


Figure 51: RF cic interval 1, mutation rate = 3/len(individual)

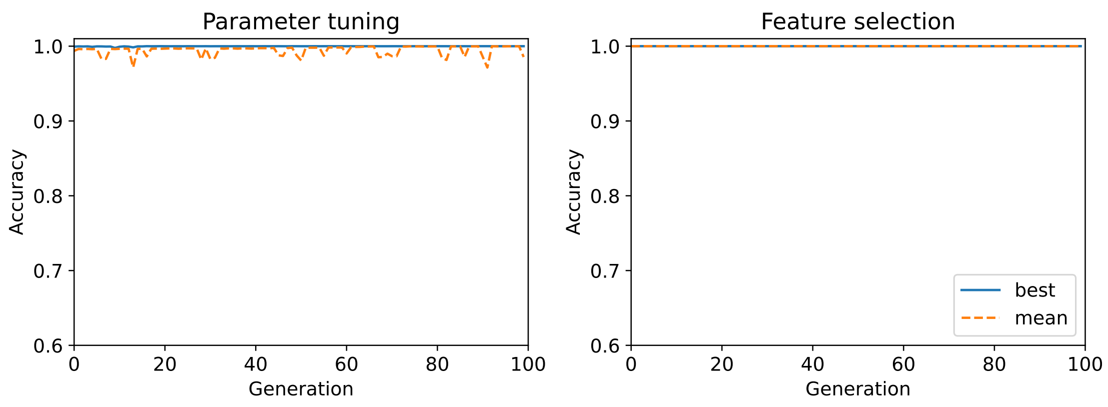


Figure 52: RF cic attacks only 10, mutation rate = 3/len(individual)

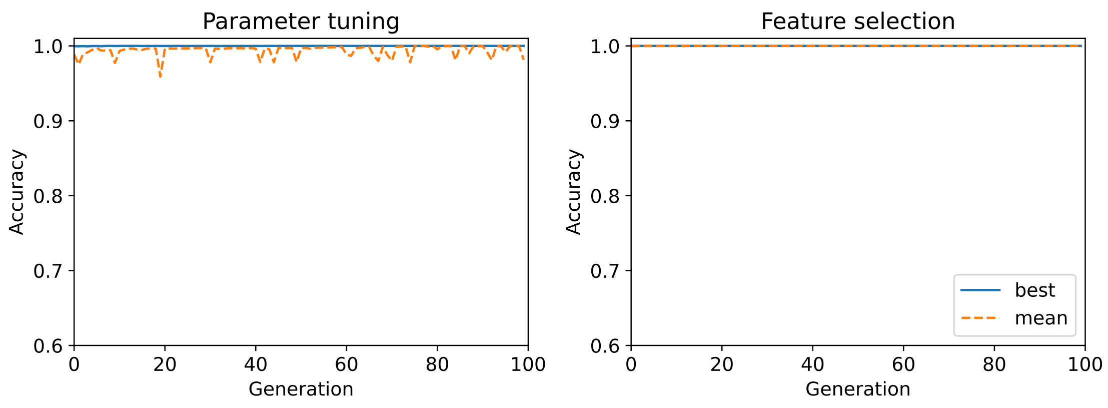


Figure 53: RF cic attacks only 1, mutation rate = 3/len(individual)

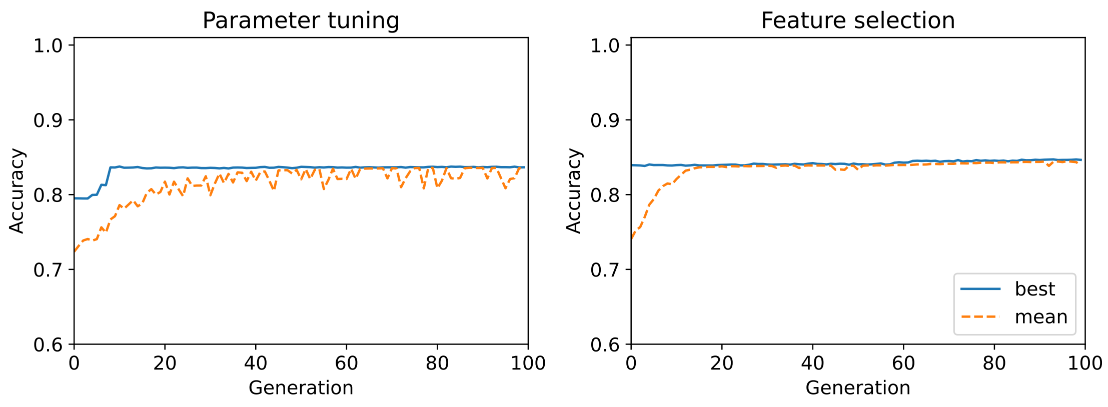


Figure 54: DT cic interval 10