

Article

Performance Study on the Use of Genetic Algorithm for Reducing Feature Dimensionality in an Embedded Intrusion Detection System

João Lobo Silva ¹ , Rui Fernandes ^{2,3,*}  and Nuno Lopes ^{2,3} 

¹ School of Technology, Polytechnic Institute of Cávado and Ave—IPCA, 4750-810 Barcelos, Portugal; a16998@alunos.ipca.pt

² 2AI-School of Technology, Polytechnic Institute of Cávado and Ave—IPCA, 4750-810 Barcelos, Portugal; nlopes@ipca.pt

³ LASI—Associate Laboratory of Intelligent Systems, 4804-533 Guimarães, Portugal

* Correspondence: rpfernandes@ipca.pt

Abstract: Intrusion Detection Systems play a crucial role in a network. They can detect different network attacks and raise warnings on them. Machine Learning-based IDSs are trained on datasets that, due to the context, are inherently large, since they can contain network traffic from different time periods and often include a large number of features. In this paper, we present two contributions: the study of the importance of Feature Selection when using an IDS dataset, while striking a balance between performance and the number of features; and the study of the feasibility of using a low-capacity device, the Nvidia Jetson Nano, to implement an IDS. The results, comparing the GA with other well-known techniques in Feature Selection and Dimensionality Reduction, show that the GA has the best F1-score of 76%, among all feature/dimension sizes. Although the processing time to find the optimal set of features surpasses other methods, we observed that the reduction in the number of features decreases the GA processing time without a significant impact on the F1-score. The Jetson Nano allows the classification of network traffic with an overhead of 10 times in comparison to a traditional server, paving the way to a near real-time GA-based embedded IDS.

Keywords: feature selection; dimensionality reduction; genetic algorithms; machine learning; HIKARI-2021



Citation: Silva, J.L.; Fernandes, R.; Lopes, N. Performance Study on the Use of Genetic Algorithm for Reducing Feature Dimensionality in an Embedded Intrusion Detection System. *Systems* **2024**, *12*, 243. <https://doi.org/10.3390/systems12070243>

Academic Editor: Zhan Shu

Received: 1 May 2024

Revised: 28 June 2024

Accepted: 4 July 2024

Published: 8 July 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In order to keep network systems operational and secure from Internet attacks, Cybersecurity defensive systems are responsible for producing and processing a sizable amount of data from firewalls, Intrusion Detection Systems (IDSs), or Intrusion Prevention Systems (IPSs) in the form of network traffic metadata and system logs [1].

IDSs are crucial in the network infrastructure as they provide a security layer that can detect a potential attack going on by analysing the network traffic in real time that violates the security policy and compromises its Confidentiality, Integrity and Availability [2–4].

IDSs are categorized into three dimensions according to Jamalipour and Mourali [5]: Intrusion Data Sources, Detection Techniques and Placement Strategies. Regarding Intrusion Data Sources, we consider a Network-based IDS (NIDS) which is used to analyse network traffic and find attacks [6]. Regarding the Detection Techniques, we consider a specification-based IDS based on rules and thresholds specified by experts, where the rules are defined by the ML model and the classification of packets is supervised by experts. Finally, regarding the Placement Strategy, we consider the use of an embedded device as a distributed IDS [7]. We also consider the use of a centralized IDS to cooperate with embedded devices, proposing a hybrid IDS.

Due to the rapid evolution of Cybersecurity threats, IDSs can quickly become outdated as new exploits and attacks emerge [8]. This ongoing challenge requires systems that are

adaptive and capable of continuous learning. Consequently, Machine Learning (ML) has become essential in this field. ML-based Intrusion Detection Systems have been developed and are showing promising results in classifying network traffic effectively [9,10].

The ability to distinguish between benign and malicious (attack) packets of the ML algorithms depends heavily on the availability of historical data on attack types and their characteristics. In the case of NIDSs, there are already various well-known datasets that cover a wide range of attack types, including CICIDS-2017 [11], KDD [12], NSL-KDD [13], UNSW-NB15 [14] and the most recent, HIKARI-2021 [15]. The network traffic-related datasets contain a lot of entries and features. The metadata from a single packet moving across the network can be quite large, and the majority of the datasets start with 50 features or more, some with more than 500,000 entries.

The number and the types of features in the dataset will make a significant impact on the time needed to train and to infer the results of the model when predicting. Feature Selection has many benefits for the models such as the prevention of overfitting, the prevention of noise and the predictive performance improvement. Some studies show that a lower number of features can increase the detection rate [16].

Genetic Algorithms have garnered significant attention from the intrusion detection research community due to their close alignment with the requirements for building efficient Intrusion Detection Systems. The high re-trainability of Genetic Algorithms enhances the adaptability of Intrusion Detection Systems, while their ability to evolve over time through crossover and mutation makes them a suitable choice for dynamic rule generation [17].

In this research, we study the impact of Feature Selection using a Genetic Algorithm in the HIKARI-2021 [15] dataset. We compare the performance of the GA with another Feature Selection technique, the chi-square test, and other Dimensionality Reduction techniques, Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), for determining the features/dimensions to be fed into the ML classifier algorithm.

The main goal of this research is to evaluate the impact of different Feature Selection/Dimensionality Reduction techniques on the performance metrics of the ML classifier. To accomplish this, we use the XGBoost algorithm, which uses the decision-tree-based predictive model through gradient boosting frameworks, as the ML classifier to detect benign and malign packets within the network dataset [18,19].

The second goal of this research is the feasibility study of an embedded system deployment as an ML-based IDS. The use of an embedded system for this context is to understand the feasibility of using a lower-power and low-budget device to protect networks from attacks using a pre-trained ML algorithm. We use the Nvidia Jetson Nano as the embedded system hardware to support the AI-based IDS running with the subset of features chosen by our Genetic Algorithm and classifying packets with the XGboost-trained model. Finally, we compare the processing time required to build a model and to predict a test set between the Jetson Nano and a traditional server.

The paper is organized as follows: Section 2 presents the Related Work discussing similar approaches from previous research; Section 3 describes the Materials and Methods in which we conduct a deep analysis of the dataset used and we detail the Feature Selection and Dimensionality Reduction algorithms; Section 4, Results and Discussion, shows a comparison between the results of Feature Selection and Dimensionality Reduction, and Section 5 concludes our work presenting the main outcomes and future work.

2. Related Work

Different algorithms for supervised learning have already been tested in the HIKARI-2021 dataset such as Gradient Boosting Machine (GBM), Light GBM, CatBoost and XGBoost [20], Multi-layer Perceptron [21], Graph Neural Networks (GNNs) [22], and other traditional ML algorithms such as KNN, Random Forest and Support Vector Machine (SVM) [23]. Although some of them achieve great results and interesting findings, most of the studies mention the importance of further research in the dataset by using different

models [24]. Different Feature Selection techniques, specifically Genetic Algorithms, have already proved some value for this case, as it is very common to have datasets with a large number of features in IDSs.

For the same purpose as this research, building an AI-based ID, Ajeet Rai studied the impact and optimization of Ensemble Methods and Deep Neural Networks by implementing Feature Selection using Genetic Algorithms, reducing from 122 to 43 features [25], and their results show that three in a total of four algorithms have better results when using Feature Selection, where the algorithm Deep Neural Network, after Feature Selection, increased its accuracy by 9%.

Those results were made in a small sub-dataset of the original KDD which probably indicates that, by using more data, the results could be even better.

On a different dataset, NSL-KDD [13], the researchers [26] tested three known algorithms for Feature Selection—Correlation-based Feature Selection (CFS) using a Genetic Algorithm, Information Gain (IG) and Correlation Attribute Evaluator (CAE)—to compare the performance of a Naive Bayes and J48 model. They also concluded that the Genetic Algorithm outperforms the others in the score while reducing the time complexity and selecting 5 out of 41 features reaching 9% more accuracy compared to the 90% they had before Feature Selection for the Naive Bayes algorithm.

The researchers in [16] used the UNSW-NB15 dataset to compare different Dimensionality Reduction techniques: Particle Swarm Optimization (PSO), Grey Wolf Optimizer (GWO), Firefly optimization (FFA) and Genetic Algorithm (GA). Their results showed that the PSO achieved better accuracy and precision than the others for the algorithms J48 and SVM, specifically, 2% more compared to the GA, by selecting 25 features out of 45 and the GA selecting 23.

Another interesting finding in this study was that it created new subsets from each subset of all individual algorithms and tested different combinations of features based on joining (inner/out) features. This achieved even better results than using just one algorithm for feature reduction.

Regarding our motivation to use embedded systems, the number of published articles shows that embedded systems are becoming popular in the field of IDSs due to their ability to be placed anywhere, the low power consumption and the cheap cost. Research shows that even an Arduino Uno can behave as an IDS using AI [27]. Another extensive study was made by comparing other low-power embedded systems with different models trained in different conditions proposing a lightweight optimized deep learning IDS [28]. Although these result shows great potential when the model starts to get bigger and the amount of data increases, there is a need for more computational power. Our approach of using the Nvidia Jetson Nano is not so common in the IDS context but it is already used in other cases such as Home Intrusions using computer vision and other techniques [29].

3. Materials and Methods

This research uses as materials the most recent publicly available IDS dataset—HIKARI-2021 [15] which is detailed in Section 3.1. The methods used in this research consist of an Exploratory Data Analysis using statistics algorithms (Section 3.1), Data Preprocessing in which we discuss the implementation and the motivation for Dimensionality Reduction and Feature Selection (Section 3.2), the use of Genetic Algorithms for Feature Selection (Section 3.3), Model Training using Machine Learning algorithms (Section 3.4) and then Model Inferencing (Section 3.5).

The system architecture that supports the materials and methods used in this research, Figure 1, comprehends five phases: Exploratory Data Analysis, where the dataset's (HIKARI-2021's) content is studied (features and data); Data Preprocessing, that applies Dimensionality Reduction and Feature Selection to the dataset; then the Model Training, using the XGBoost algorithm and the best subset of features previously selected; and, lastly, the deployment of the model to the embedded system.

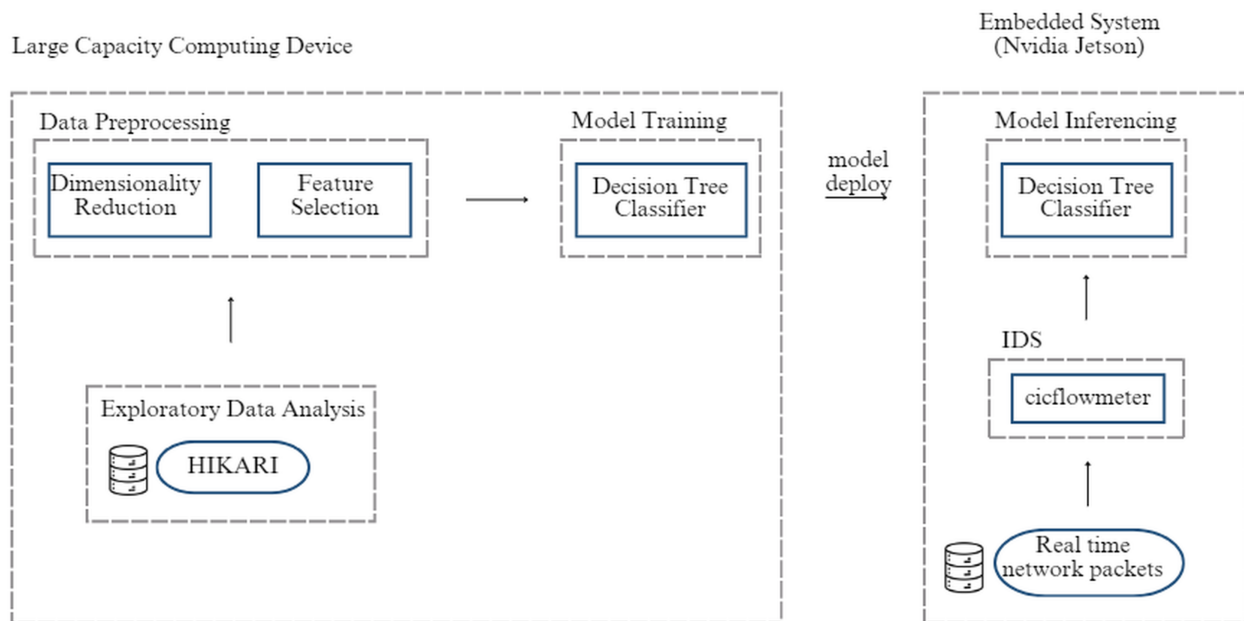


Figure 1. System architecture.

In the embedded system, the same performance evaluation is made as well as in the traditional server that ran the other experiments. This test consists of running the different models trained with different numbers of features measuring the time of inferencing the test set. The goal is to keep the classification performance and verify which runs faster and whether the embedded system could be a good approach to this use case.

3.1. Exploratory Data Analysis (EDA) of the HIKARI-2021 Dataset

The HIKARI-2021 [15] dataset is currently the most up to date, containing encrypted synthetic attacks and benign traffic. The main advantages of this dataset compared to the others is that the network logs are only anonymous in specific logs of benign traffic, the dataset can be reproducible and has ground-through data, and the main characteristic that distinguishes it from the others: having encrypted traffic. This dataset has a total of 555,278 records and 88 features, where 2 of them are labels (one binary and other string) where one corresponds to 1 if the traffic is malicious and 0 means benign or background, and another label that classifies the traffic into the different categories: Benign, Background, Probing, Bruteforce, Bruteforce-XML and XMRIGCC-CryptoMiner.

In Table 1 it is possible to see the names of the features as well as their types. Most of them are integers or floats which are related to the traffic analysis, and some are strings related to the identification of the source/destination and label.

One problem of this dataset, which is normal in datasets of IDSs, is the fact that it is unbalanced. Table 2 shows well how unbalanced the dataset is, having the greater quantity of traffic in the benign and background traffic which is normal traffic.

Based on a HIKARI-related study [30] the authors conducted a feature analysis that concluded that some of the features might be removed to accomplish better results. The same approach was followed here, removing the columns “Unnamed: 0.1”, “Unnamed: 0.2”, “uid”, “originh” and “originp” which are not relevant for a generic classification model but contribute to bias in the training step of the classification by learning specific id tags that would otherwise not appear in a real traffic dataset.

Table 1. Features names and types for HIKARI-2021 dataset.

	Feature Name	Feature Type		Feature Name	Feature Type		Feature Name	Feature Type
1	Unnamed: 0.1	int64	31	flow_CWR_flag_count	int64	61	flow_iat.avg	float64
2	Unnamed: 0	int64	32	flow_ECE_flag_count	int64	62	flow_iat.std	float64
3	uid	string	33	fwd_pkts_payload.min	float64	63	payload_bytes_per_second	float64
4	originh	string	34	fwd_pkts_payload.max	float64	64	fwd_subflow_pkts	float64
5	originp	int64	35	fwd_pkts_payload.tot	float64	65	bwd_subflow_pkts	float64
6	responh	string	36	fwd_pkts_payload.avg	float64	66	fwd_subflow_bytes	float64
7	responp	int64	37	fwd_pkts_payload.std	float64	67	bwd_subflow_bytes	float64
8	flow_duration	float64	38	bwd_pkts_payload.min	float64	68	fwd_bulk_bytes	float64
9	fwd_pkts_tot	int64	39	bwd_pkts_payload.max	float64	69	bwd_bulk_bytes	float64
10	bwd_pkts_tot	int64	40	bwd_pkts_payload.tot	float64	70	fwd_bulk_packets	float64
11	fwd_data_pkts_tot	int64	41	bwd_pkts_payload.avg	float64	71	bwd_bulk_packets	float64
12	bwd_data_pkts_tot	int64	42	bwd_pkts_payload.std	float64	72	fwd_bulk_rate	float64
13	fwd_pkts_per_sec	float64	43	flow_pkts_payload.min	float64	73	bwd_bulk_rate	float64
14	bwd_pkts_per_sec	float64	44	flow_pkts_payload.max	float64	74	active.min	float64
15	flow_pkts_per_sec	float64	45	flow_pkts_payload.tot	float64	75	active.max	float64
16	down_up_ratio	float64	46	flow_pkts_payload.avg	float64	76	active.tot	float64
17	fwd_header_size_tot	int64	47	flow_pkts_payload.std	float64	77	active.avg	float64
18	fwd_header_size_min	int64	48	fwd_iat.min	float64	78	active.std	float64
19	fwd_header_size_max	int64	49	fwd_iat.max	float64	79	idle.min	float64
20	bwd_header_size_tot	int64	50	fwd_iat.tot	float64	80	idle.max	float64
21	bwd_header_size_min	int64	51	fwd_iat.avg	float64	81	idle.tot	float64
22	bwd_header_size_max	int64	52	fwd_iat.std	float64	82	idle.avg	float64
23	flow_FIN_flag_count	int64	53	bwd_iat.min	float64	83	idle.std	float64
24	flow_SYN_flag_count	int64	54	bwd_iat.max	float64	84	fwd_init_window_size	int64
25	flow_RST_flag_count	int64	55	bwd_iat.tot	float64	85	bwd_init_window_size	int64
26	fwd_PSH_flag_count	int64	56	bwd_iat.avg	float64	86	fwd_last_window_size	int64
27	bwd_PSH_flag_count	int64	57	bwd_iat.std	float64	87	traffic_category	string
28	flow_ACK_flag_count	int64	58	flow_iat.min	float64	88	Label	int64
29	fwd_URG_flag_count	int64	59	flow_iat.max	float64			
30	bwd_URG_flag_count	int64	60	flow_iat.tot	float64			

Table 2. Distribution of the HIKARI-2021 dataset per label.

Traffic Category	Frequency
Benign	347,431
Background	170,151
Probing	23,388
Bruteforce	5884
Bruteforce-XML	5145
XMRIGCC CryptoMiner	3279

3.2. Data Preprocessing

Regarding the Data Preprocessing methods, there are a wide range of options that can be implemented, and they must be used according to the data type and the problem we are dealing with. This research, as mentioned previously, addresses the problem of IDS datasets which have a large number of features.

To address the problem of the high number of features there are two main approaches: Dimensionality Reduction and Feature Selection [31].

These techniques are particularly useful in Machine Learning and Data Analysis to address issues related to high dimensionality, multicollinearity and overfitting. The high dimensionality of IDS datasets makes them an excellent use case of these methods since we can get a better understanding of the importance of each feature, and we can also reduce the training and inference times of the models used for intrusion detection tasks.

3.2.1. Dimensionality Reduction

In the Dimensionality Reduction field, the techniques reduce the dimensionality of the dataset by transforming the original features into a new set of features, that are a mapping of the original ones, while trying to preserve the essential information and patterns present in the data. One disadvantage is that, when the original features have an obvious physical meaning, the new features may lose meaning. Methods like Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) fall into this category.

PCA reduces the complexity of high-dimensional data by identifying principal components—orthonormal eigenvector and eigenvalue pairs—where data variance is maximized. It projects the data onto these components, focusing on those with the highest eigenvalues and compressing the data while preserving essential information [32].

LDA projects the n -dimensional dataset into a smaller k -dimensional dataset ($k < n$), while at the same time maintaining all the relevant class discrimination information [33].

3.2.2. Feature Selection

Feature Selection, on the other hand, focuses on selecting a subset of the original features while discarding the rest. The selected features are considered the most relevant for the specific task, and irrelevant or redundant features are removed. Chi squared (Chi-2) and Genetic Algorithms (GA) are both examples of Feature Selection methods.

Chi squared is a numerical test that measures the deviation from the expected distribution considering the feature event is independent of the class value [34]. Its use can be as simple as importing the method from SciKit-Learn and defining a K number of top features that have more impact on the label.

A Genetic Algorithm (GA) is an optimization strategy that operates on a population of potential solutions, often represented as individuals or candidate solutions, with the use of genetic operators such as selection, crossover and mutation to evolve and improve the population over generations [35]. This type of algorithm has been around for many years and has proven itself to be a useful approach to finding solutions to real-world complex problems.

3.3. Genetic Algorithm for Feature Selection

In a GA approach to Feature Selection, each solution represents a subset of features in the dataset, and the fitness function is used to evaluate the quality of the solutions. The fitness function measures the accuracy of the model trained on each solution, with higher accuracy indicating a better set of features.

3.3.1. Genetic Encoding and Initialization

The process starts by identifying a way to express our solution as a genome.

For Feature Selection, the genome is a binary one-dimensional array with the same length as the number of features in the HIKARI-2021 dataset (Figure 2), where each bit represents the removal or retention of a specific feature (Table 3).

[1,1,1,1,0,1,0,1]

Figure 2. Genetic encoding of dataset.

Table 3. Features in array.

Features	Removed
Feature 1	No
Feature 2	No
Feature 3	No
Feature 4	No
Feature 5	Yes
Feature 6	No
Feature 7	Yes
Feature 8	No

The subsequent step in the GA is to randomly initialize a population of solutions, where each solution represents a subset of features.

In our experiments, we used an initial population of 25. This randomness in initialization is critical because it ensures a diverse set of starting points for the Genetic Algorithm. A diverse initial population helps prevent premature convergence to sub-optimal solutions and allows the algorithm to explore a wide variety of potential solutions.

The next step in the GA is to randomly generate an initial population of possible solutions where each one is a combination of features. This randomness in initialization is essential because it guarantees a diverse set of starting points for the algorithm. This diverse starter population ensures that the algorithm does not get trapped in local optima and is able to search through many possible sets of solutions.

The sparsity of each individual solution or, in this case, the number of features to be removed needs to be chosen beforehand, so that the algorithm will generate arrays with the correct number of features. Each individual is applied to the dataset as a sort of mask, where if a feature is set as “0” in the array, it will be removed from the dataset. After going through the entire array, and removing or maintaining features, the resulting dataset is ready to be tested.

3.3.2. Selection Method

This population is evaluated through a fitness score.

In this case, the score of each individual (dataset) is the accuracy of the Machine Learning model when trained with only the newly selected features; this also means that we train the model multiple times depending on how many individuals there are in the population. The model’s fitness score is stored so it can be later compared to the other individuals in that generation.

The algorithm will repeat this process of evaluating the fitness of each solution. After it has evaluated every individual in that population, the next step is to choose the fittest individuals to pass their genes onto the next generation.

Every individual has a chance of becoming a parent with a probability proportional to its fitness.

The fitter individuals have a better chance of breeding and passing on their genes to the next generation.

The selection method was a memory-optimized version of the roulette wheel approach [36]. Inside the selection function, an individual’s fitness is compared to a randomly

generated float value between 0 and 1; if the fitness is high, there is a larger probability that the random number will be less than the fitness, resulting in the individual being chosen as a parent.

3.3.3. Crossover and Mutation

The crossover phase of the algorithm is when two selected individuals (parents) are combined to produce new offspring. This phase is crucial because it enables the mixing of genetic material from different parents [37].

Two examples of crossover techniques are as follows:

1. Single-Point Crossover: Here, a single point is chosen at random along the length of the parent chromosomes, splitting the chromosomes (Figure 3). The resulting segments are then swapped between the parents generating two new solutions (children).
2. Multi-Point Crossover: As the name suggests, multiple points are selected and segments between these points are swapped (Figure 4).



Figure 3. Single-Point Crossover.

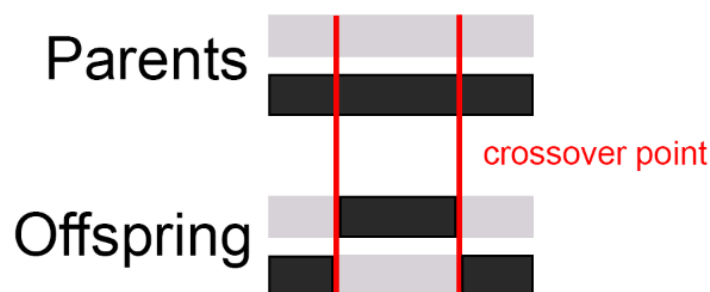


Figure 4. Multiple-Point Crossover.

When a new generation of offspring is born, some of their genes may be susceptible to a low-probability mutation. This means that some of the bits in the genetic code can be switched around. Mutations are an important step of the GA because it guarantees diversity in the population and prevents premature convergence. Individuals with a high fitness score would soon dominate the population in the absence of mutations, limiting the GA to a local optimum and hindering the GA's ability to produce superior offspring.

In our experiments, Single-Point Crossover was utilized and the point would be chosen at random. After the new individuals are formed before they are sent into the next generation, a mutation probability of 1% was applied to every gene in the solution. This means that the algorithm will parse through the solution and there is a 1% chance of that bit being altered.

3.3.4. Sparsity Check

Removing features from the dataset, most of the time, results in a loss in accuracy, especially when random features are dropped. This indicates that a genome composed of a

lot of features will probably have better accuracy than a genome with almost no features so the GA will converge on solutions where almost no features are removed but, since we are aiming to reduce the dataset's dimensions, there must be a mechanism to control the sparsity levels of genome.

To address this issue, a sparsity check function was built to choose the pruning percentage which one wants to apply to the model, as well as to ensure that the generated offspring meet these criteria; if they do not, an algorithm randomly selects values in the mask to modify, to meet the sparsity criteria.

3.4. Model Training

Our model was based on the XGBoost Classifier library for Python and training was conducted using all parameters set to default [38].

Regarding the categorical features, the SciKit-Learn label encoder was used that consists of transforming the categories in an integer creating a dictionary of pair categories due to a requirement of the XGBoost algorithm due to the efficiency of the algorithm and the numerical computation needed for decision trees.

A crucial way to be confident when saying a model is ready to be used and fits the expectations of the job needed is by evaluating the model. Different metrics can be used; each problem will require its own metrics regarding speed, performance, results, outcomes and other things. For this specific topic and because the focus was the performance of classification, it was decided to use the most popular metrics in IDS: accuracy and *F1-score*.

Accuracy is calculated by dividing the number of correctly identified predictions, true positives (TP) and true negatives (TN) by the total number of predictions (TP + FN + TN + FP) which means that by having a high accuracy the algorithm can be more trusted to classify the traffic but in IDS field accuracy is not the most important thing because the existing datasets are imbalanced and the model can label everything as "benign" and achieve high accuracy due to the fact that the label with the highest frequency is "benign"; balanced accuracy is a better option for this case.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (1)$$

Precision is calculated by dividing the number of correctly identified malicious predictions, true positives (TP), by the total number of predictions that the algorithm classified as malicious, true positives (TP) + false positive (FP), which means that high precision is valuable to say that we can correctly identify the malicious traffic with a low percentage of false positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

Recall, also known as sensitivity or true positive (TP) rate, is a metric that focuses on the ability of a model to correctly identify all positive instances in the dataset. It is calculated by dividing the true positives (TP) by the sum of true positives (TP) and false negatives (FN). In the IDS context, recall is a crucial metric since it represents the ability of a model to catch as much malicious traffic as possible. High recall means that the model is able to identify the majority of true threats; however, it may lead to more false positives (FP).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

The F1-score is a metric that combines both precision and recall providing a balanced measure of a model's performance. It is useful when it is required to find a balance between precision and recall, ensuring that false positives and false negatives are both minimized. The F1-score is the metric that is most trustworthy in IDS studies because when it is higher it ensures that the model not only catches malign traffic but also minimizes false positives.

$$F1\ score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

3.5. Model Inferencing on the IDS

Regarding the inference, the goal is to combine multiple parts into a single fully working embedded IDS.

The system receives as input the trained model (transferred as a “.pkl” file) and the selected feature’s identification to analyse in the captured packets. Every IDS system requires the capture of network packets. Two options are available, either the capture is made in real-time, using a packet sniffer, or the capture uses an offline network packet file (like “.pcap”). In our case, we used the Cicflowmeter, which is a publicly available packet sniffer, that captures network traffic and loads it into our inferencing model. The inference algorithm must load each packet to be analysed and, from it, it must extract only the selected features, to be processed in the trained model.

As an output, our IDS identifies packets which are either benign or malign, and, within the malign classification, our model identifies the attack type.

All of these parts fit into an embedded system, the Nvidia Jetson Nano, which runs a Linux Ubuntu distribution with both Cicflowmeter and our inference algorithm for real-time predictions. We used the Nvidia Jetson Nano Developer Kit, which has a Quad-core ARM A57 @ 1.43 GHz CPU, 128-core Maxwell GPU and 4 GB LPDDR4 as memory. This device is specially designed to run Neural Networks and ML algorithms [39].

Finally, in order to have a working IDS, the network environment must be configured to have all the inbound and outbound packets sent to the embedded device. In this way, all network packets are analysed by our device, working as an IDS.

4. Results and Discussion

In this final section, we dive into a comparative analysis of Dimensionality Reduction techniques applied to our dataset in conjunction with the XGBoost model. Specifically, we explore the impact of Dimensionality Reduction using Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA), as well as Feature Selection through chi squared (Chi-2) and the Genetic Algorithm.

Our goal is to understand the effectiveness of each technique in reducing the dimensions of the HIKARI-2021 dataset while striking a balance with the XGBoost model’s performance.

This reduction comes with a trade-off of reduction in the values of our metrics, for example, 0.4% of F1-score when reducing from 83 to 20 features.

The server used was a single machine with a dual Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz 10 core CPU, with 120 GB of RAM with no powerful background apps running simultaneously.

4.1. Dimensionality Reduction

As previously mentioned, the two Dimensionality Reduction techniques utilized were PCA and LDA. With PCA, it was possible to choose between 70 and 1 features. But with LDA it was limited to a maximum of five feature dimensions (Table 4).

LDA is a supervised method that aims to find the directions (linear discriminants) that maximize the separation between different classes. The number of linear discriminants is at most $c-1$, where c is the number of classes. This is because LDA projects data points onto a lower-dimensional space to maximize class separability, and the maximum number of discriminative directions is determined by the number of classes [40].

On the other hand, PCA is an unsupervised method that does not take into account class labels. It finds the directions (principal components) that maximize the variance of the data. Therefore, PCA can generate as many principal components as the original dimension of the feature space or the number of samples, whichever is smaller.

Table 4. LDA.

Feature Dimensions	Train Time (s)	Test Time (s)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score Weighted (%)
5	94.22	0.065	72.22	70.59	69.64	70.11
4	93.51	0.064	71.64	69.96	68.93	69.45
3	84.78	0.056	70.67	68.79	67.60	68.19
2	76.96	0.066	69.65	67.17	66.22	66.69
1	61.03	0.074	66.77	61.62	65.65	63.58

Regarding LDA's effectiveness, from the experimental results, we can see that it achieved lower accuracy and a lower F1-score for the same dimensions as PCA. This shows that PCA can strike a better balance between feature dimension and performance.

In both approaches, Tables 4 and 5, there is a noticeable decrease in training time as the feature dimension is also reduced, but there is no difference in the inference times on the test set; this is due to the high computational power of the computer used, where the number of data on the test set is so small that it does not make too much of an impact.

Table 5. PCA.

Feature Dimensions	Train Time (s)	Test Time (s)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score Weighted (%)
70	217.12	0.073	76.53	75.03	75.44	75.23
60	210.09	0.073	76.43	74.84	75.39	75.13
50	192.97	0.068	76.32	74.79	75.17	74.98
40	120.18	0.073	75.54	74.03	74.38	74.20
30	116.52	0.068	75.25	73.92	73.85	73.88
20	111.14	0.064	74.22	72.77	72.56	72.66
10	94.85	0.066	73.71	72.27	72.59	71.93
5	83.03	0.063	73.11	71.34	70.67	71.00
4	77.33	0.072	72.43	70.78	69.58	70.17
3	77.98	0.071	71.88	68.30	70.74	69.49
2	60.25	0.073	71.45	69.39	42.65	69.54
1	46.99	0.071	69.79	65.40	66.11	65.75

4.2. Feature Selection

Concerning the Feature Selection techniques, we can see that both Chi-2 and the GA achieved better results than the Dimensionality Reduction of LDA; both Chi-2 (Table 6) and the GA (Table 7) maintained a higher accuracy and F1-score as the feature dimensions lowered. The GA was the one with the better results; it achieved 71.00% F1-score with only a dimension of 1, beating the PCA, the LDA and the Chi-2.

The lower results of the Chi-2 compared to the GA can be explained by its approach to Feature Selection since it relies on statistical measures to identify the relationship between each feature and the target variable. However, it may not explore the entire feature space comprehensively, and its effectiveness can be limited by the relationship between features. The GA, on the other hand, performs a more exhaustive search of feature combinations through its evolutionary process, making it more likely to discover relevant feature interactions and dependencies.

One of the main drawbacks of the GA is its runtime, as in the total time that the Genetic Algorithm took to go through the various possible candidates in each generation. In Table 7, we can see that it takes a considerable time to find the optimal solution for the different feature dimensions that are selected; this time decreases as the number of dimensions also becomes smaller but it is an important thing to consider, since the other approaches to reduce the dimensions are almost instantaneous.

Table 6. Chi-2.

Feature Dimensions	Train Time (s)	Test Time (s)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score Weighted (%)
70	101.5	0.097	77.91	76.40	76.80	76.60
60	87.83	0.093	77.93	76.55	76.71	76.63
50	77.21	0.088	77.89	76.52	76.74	76.62
40	78.72	0.082	77.56	76.19	76.18	76.19
30	79.13	0.083	76.36	75.17	74.44	74.80
20	80.31	0.072	73.85	72.63	71.60	72.11
10	68.98	0.063	73.23	71.86	70.83	71.34
5	64.35	0.063	72.35	70.48	69.38	69.92
4	66.42	0.078	71.93	70.03	68.84	69.43
3	52.02	0.063	70.56	68.30	66.63	67.45
2	22.8	0.065	63.91	61.43	46.34	52.83
1	17.09	0.074	62.89	56.11	44.69	49.75

Table 7. Genetic Algorithm.

Feature Dimensions	GA Run Time (m)	Average Train Time per Individual and per Generation (s)	Average Test Time per Individual and per Generation (s)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score Weighted (%)
70	901.9	96.58	0.096	78.08	76.59	76.97	76.78
60	897.6	95.83	0.101	78.09	76.60	77.00	76.80
50	807.0	90.35	0.09	78.07	76.63	76.92	76.77
40	702.5	71.05	0.079	78.16	76.82	76.98	76.89
30	702.0	75.53	0.073	77.96	76.54	76.76	76.64
20	709.6	76.02	0.082	77.83	76.46	76.70	76.57
10	606.9	68.63	0.066	77.42	76.14	76.14	76.14
5	476.1	58.8	0.065	75.99	74.88	74.42	74.65
4	563.9	65.97	0.064	76.09	74.70	74.48	74.59
3	509.2	61.58	0.062	75.16	73.59	73.21	73.40
2	351.3	27.32	0.06	73.56	71.00	71.19	71.09
1	295.5	26.8	0.055	71.42	68.92	71.16	71.00

This time does not negatively impact the final system because the algorithm runs only once to select the best subset of features. When classifying packets, the algorithm will take the same amount of time as other algorithms because the number of features is the same. The main impact is in the number of features; by selecting fewer features, the algorithm will run faster.

With these two methods, we can see which features are chosen as the most impactful to the accuracy of the model. On PCA and LDA, this was not possible since the features were transformed. In Tables 8 and 9 the five most important features are shown.

Table 8. The 5 most important features according to Chi-2.

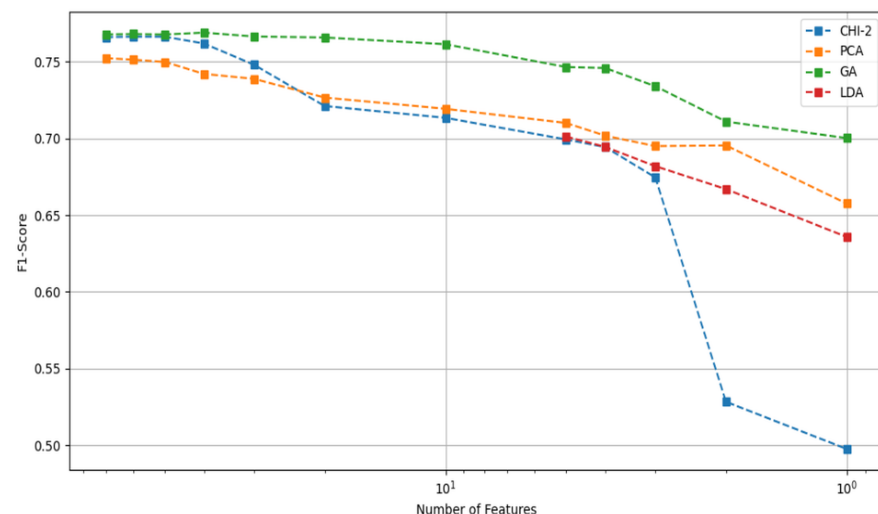
Feature Name
bwd_bulk_rate
idle.tot
flow_iat.tot
fwd_iat.tot
bwd_iat.tot

Table 9. The 5 most important features according to GA.

Feature Name
bwd_pkts_payload.max
fwd_bulk_packets
active.avg
idle.avg
fwd_last_window_size

The selected features of chi squared and Genetic Algorithms are similar regarding their value. All of them are related to timestamps of packets sent or received, or time of inactivity. Related to the GA features selected, feature “bwd_pkts_payload.max” is the maximum payload size of packets in the backward direction which allows us to understand whether large amounts of data are being sent out from the network; “fwd_bulk_packets” is the number of bulk packets in the forward direction which helps in understanding the volume of data being sent from the source which can indicate unauthorized data transfer; “active.avg” is the average time of the flow providing a measure of the typical duration which can indicate whether the connections are too short or too long; “idle.avg” is the average inactivity period within flows which can be suspect if this time is extended; “fwd_last_window_size” is the size of the last windows in the forward direction and can indicate the termination patterns of network sessions and whether irregularity can be suspicious.

When comparing the results of all the techniques used, we can visualize in Figure 5 that the GA emerged as the most effective approach in maintaining optimal performance even as the number of features was decreased. In contrast, the Chi-2 approach demonstrated a poorer performance when compared to its counterparts.

**Figure 5.** Number of features vs. F1-score (XX axis is in log scale).

In Tables 10 and 11, we present a summary overview of the metrics indicators for the four algorithms for Dimensionality Reduction and Feature Selection. For each algorithm, we selected the best F1-score result, considering all the different number of feature cases. As observed the GA presented the best F1-score result, with 40 features achieving 76.89% of the F1-score as shown in Table 10.

In Table 11, we compared the same algorithms with the same metrics but using the same number of features/dimensions—the top five. As observed, the GA presented the best F1-score result, with five features achieving 74.65% of the F1-score.

Table 10. Metrics indicators for the best subset of features/dimensions according to F1-score.

	Number of Features	Accuracy	Precision	Recall	F1-Score
PCA	70	76.53	75.03	75.44	75.23
LDA	5	72.22	70.59	69.64	70.11
Chi squared	60	77.93	76.55	76.71	76.63
Genetic Algorithm	40	78.16	76.82	76.98	76.89

Table 11. Metrics indicators for the top 5 features/dimensions.

	Number of Features	Accuracy	Precision	Recall	F1-Score
PCA	5	73.11	71.34	70.67	71.00
LDA	5	72.22	70.59	69.64	70.11
Chi squared	5	72.35	70.48	69.38	69.92
Genetic Algorithm	5	75.99	74.88	74.42	74.65

4.3. Performance Evaluation of the Jetson Nano

One of the goals of this paper is to assess the feasibility of using an embedded system to implement the IDS functionality. With that in mind, we evaluate all the different models, trained in the server machine, with the same reference test set in the Jetson device and compare the time needed to infer the complete test set. The results in Table 12 show that the Nvidia Jetson takes 10 times longer than the server to classify the traffic in the test set—about 111,000 entries.

Table 12. Inference time in the Jetson Nano vs in the server.

# Features	Jetson Nano				Server			
	Chi Squared	GA	LDA	PCA	Chi Squared	GA	LDA	PCA
1	0.7151	0.6798	0.8331	0.7814	0.0740	0.0553	0.0744	0.0714
2	0.7552	0.7275	0.7405	0.7703	0.0647	0.0604	0.0662	0.0729
3	0.7670	0.7275	0.7201	0.7811	0.0632	0.0616	0.0568	0.0707
4	0.7538	0.7320	0.7394	0.7847	0.0776	0.0635	0.0640	0.0724
5	0.7597	0.7449	0.7392	0.7717	0.0630	0.0654	0.0650	0.0629
10	0.7508	0.7164		0.7926	0.0632	0.0661		0.0663
20	0.8356	0.7647		0.8152	0.0722	0.0818		0.0643
30	0.8336	0.8235		0.8705	0.0827	0.0732		0.0675
40	0.8481	0.8484		0.8871	0.0817	0.0792		0.0730
50	0.8494	0.8683		0.8852	0.0879	0.0903		0.0675
60	0.9322	0.9375		0.9483	0.0931	0.1014		0.0734
70	0.9861	0.9345		0.9602	0.0969	0.0959		0.0732

From the results table, it is noticed that, although chi squared is the algorithm that has a higher F1-score, it is also the one that takes more time, both in the Jetson and in the server, to predict the test set. Another relevant aspect is that LDA takes more time to predict the complete test set when configured to use one component than when using five components.

The results can be graphically explained in Figure 6 where the xx axis identifies the number of features considered and the yy axis (which is in a log scale) represents the time used in seconds for running the inference on the test set.

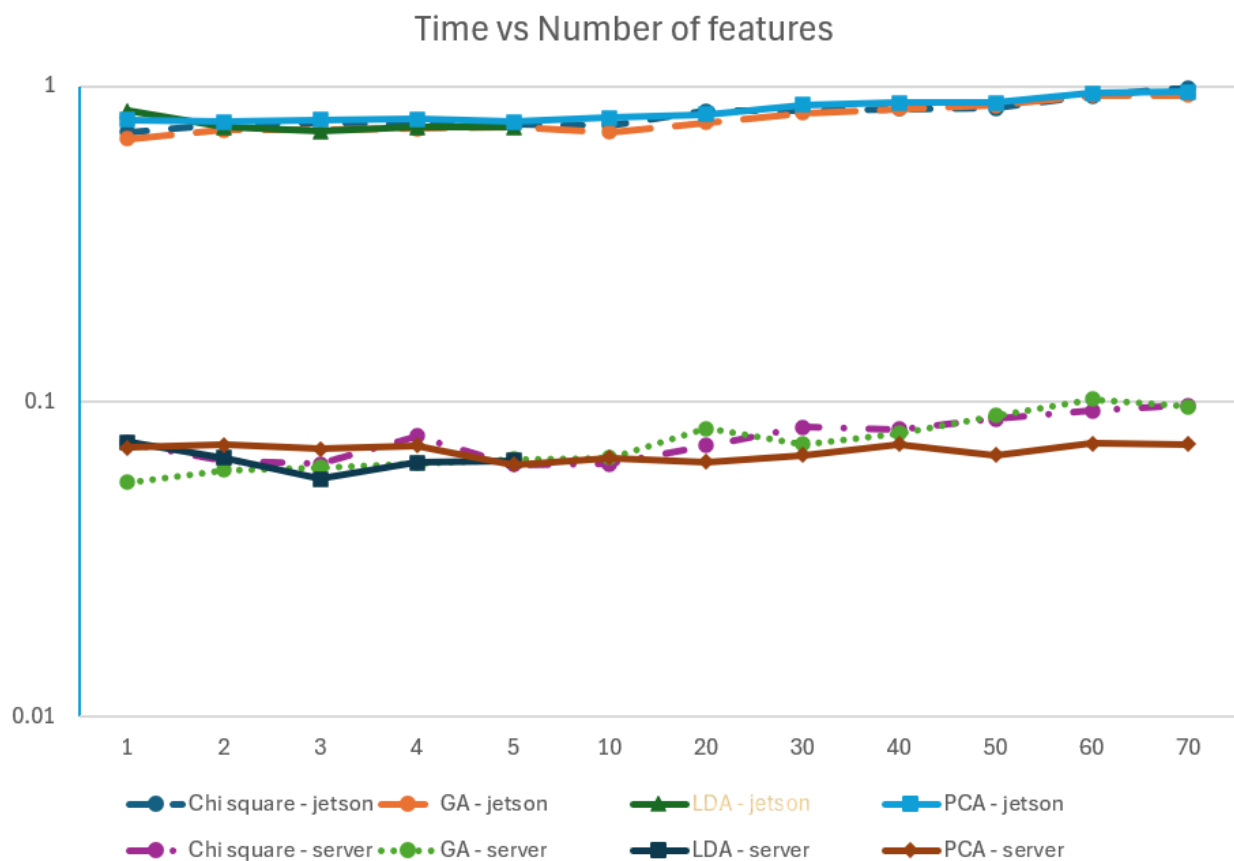


Figure 6. Time vs. number of features (YY axis is in log scale).

We can observe that the time needed to infer in the Jetson Nano is 10 times longer than the time needed by the server. It is also visible, both for the server and for the Jetson, that the time taken for the inference increases with the number of features, which is explainable by the complexity of having more features to process.

It is important to note that these tests were made in a controlled environment with a specific set of network traffic (dataset) and other future tests must be conducted in real time to validate the proposed system.

This latency of predicting time in the Jetson is an identified constraint that cannot easily be bypassed. The embedded devices, which are part of the hybrid IDS placement architecture [41], can detect some malicious packets based on their current ML model. In the case where malicious packets are not detected by the embedded devices, the devices can be configured to send packet captures to the central server, so that this server can perform a more in-depth analysis afterwards. From this analysis, the central server can update the ML model and transfer it to the embedded devices in a quick manner (which would be equivalent to a ".pkl" file transfer).

Figure 7, also with the xx axis representing the number of features and the yy axis representing the inference time in seconds in a log scale, represents the time of the four models on the Jetson device. It is easily noticed that the increase in the number of features also increases the time to infer. The exception is the LDA algorithm, where running one component is slower than running with five components.

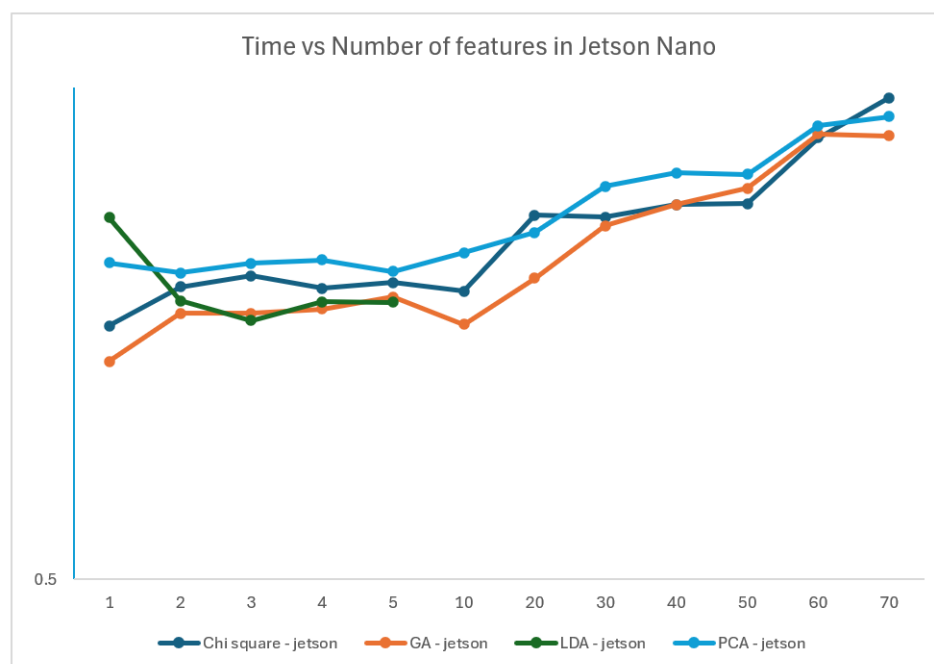


Figure 7. Time vs. number of features in the Jetson Nano (YY axis is in log scale).

5. Conclusions

In this work, we study the importance of Feature Selection when using an IDS dataset. First, an exploratory data analysis was conducted in the HIKARI-2021 dataset to characterize the features. The dataset contains four attack types and two benign types. The distribution ratio between benign and malicious packets is 93% to 7%.

We then performed Data Preprocessing to reduce the model complexity. We propose to reduce the number of features considered, to achieve the same performance metrics (F1-score) but with a faster time to run the model. We compared two different approaches: Dimensionality Reduction and Feature Selection. For the Dimensionality Reduction, we used the PCA and the LDA algorithms. For the Feature Selection, we used the chi-squared test and the Genetic Algorithm. We applied these algorithms to the HIKARI-2021 dataset and evaluated the performance metrics and running time.

The results show that the Genetic Algorithm outperformed the most traditional algorithms such as PCA, LDA and chi-square test, according to the F1-score, for all feature/dimension sizes. The GA obtained an F1-score of 76.89% for 40 features while the chi squared had 76.63% for 60 features, the PCA obtained 75.23% for 70 features and the LDA obtained 70.11% for 5 features. Although the GA presents better results regarding the F1-score and with a smaller number of features, the gain was marginally small, except for the LDA which uses a much smaller number of dimensions. The GA requires more processing time than the others to select the most relevant features; however, we observed that the reduction in the number of features decreases the GA processing time without a significant impact on the F1-score.

We have compared all algorithms for the same number (five) of features/dimensions. In this case, the results also showed that the GA provides the best F1-score, 74.65% when compared to the PCA with 71.00%, the LDA with 70.11% and the chi-square with 69.92%.

We also studied the use of an embedded system, the Nvidia Jetson Nano, to implement the IDS functionality as an edge device. A comparison was made between this device and a typical server to evaluate this system. The results show that the Jetson can behave as an IDS but with an increase by a factor of 10 on the inference time when compared to the server.

As future work, we plan to deepen the study of the algorithm selection by conducting statistical tests to validate the significance of the different performances of the multiple algorithms tested for identifying feature relevance. We have only used the classic ML

metrics to measure the classification. Additionally, we intend to compare the classification performance when considering other well-known datasets, such as CICIDS-2017 and KDD99. In this study, we have used default configuration parameters. We can also study the tuning of hyper-parameters to improve the overall algorithm performance. Finally, our embedded system study was not tested on a real-time capture and inference environment. We propose to validate our embedded system as a real-time IDS implementation and compare it to the current state of the art.

Author Contributions: Conceptualization, J.L.S., R.F. and N.L.; methodology, J.L.S., R.F. and N.L.; investigation, J.L.S., R.F. and N.L.; writing, J.L.S., R.F. and N.L.; supervision, N.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Mohammad, R.; Saeed, F.; Almazroi, A.A.; Alsubaei, F.S.; Almazroi, A.A. Enhancing Intrusion Detection Systems Using a Deep Learning and Data Augmentation Approach. *Systems* **2024**, *12*, 79. [CrossRef]
2. Vasilomanolakis, E.; Karuppayah, S.; Mühlhäuser, M.; Fischer, M. Taxonomy and Survey of Collaborative Intrusion Detection. *ACM Comput. Surv.* **2015**, *47*, 1–33. [CrossRef]
3. Denning, D. An Intrusion-Detection Model. *IEEE Trans. Softw. Eng.* **1987**, *SE-13*, 222–232. [CrossRef]
4. Alsajri, A.; Steiti, A. Intrusion Detection System Based on Machine Learning Algorithms: (SVM and Genetic Algorithm). *Babylon. J. Mach. Learn.* **2024**, *2024*, 15–29. [CrossRef] [PubMed]
5. Jamalipour, A.; Murali, S. A Taxonomy of Machine-Learning-Based Intrusion Detection Systems for the Internet of Things: A Survey. *IEEE Internet Things J.* **2022**, *9*, 9444–9466. [CrossRef]
6. Samrin, R.; Vasumathi, D. Review on anomaly based network intrusion detection system. In Proceedings of the 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, India, 15–16 December 2017; pp. 141–147. [CrossRef]
7. Tlili, F.; Ayed, S.; Chaari Fourati, L. Exhaustive distributed intrusion detection system for UAVs attacks detection and security enforcement (E-DIDS). *Comput. Secur.* **2024**, *142*, 103878. [CrossRef]
8. Oliveira, R.; Pedrosa, T.; Rufino, J.; Lopes, R.P. Parameterization and Performance Analysis of a Scalable, near Real-Time Packet Capturing Platform. *Systems* **2024**, *12*, 126. [CrossRef]
9. Halimaa A., A.; Sundarakantham, K. Machine Learning Based Intrusion Detection System. In Proceedings of the 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 23–25 April 2019; pp. 916–920. [CrossRef]
10. Ahmad, Z.; Shahid Khan, A.; Shiang, C.; Ahmad, F. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Trans. Emerg. Telecommun. Technol.* **2021**, *32*, e4150. [CrossRef]
11. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In Proceedings of the International Conference on Information Systems Security and Privacy, Madeira, Portugal, 22–24 January 2018.
12. KDD Cup 1999 Data. Available online: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> (accessed on 3 July 2024)
13. Tavallae, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the KDD CUP 99 data set. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009; pp. 1–6. [CrossRef]
14. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, Australia, 10–12 November 2015; pp. 1–6. [CrossRef]
15. Ferriyan, A.; Thamrin, A.H.; Takeda, K.; Murai, J. Generating Network Intrusion Detection Dataset Based on Real and Encrypted Synthetic Attack Traffic. *Appl. Sci.* **2021**, *11*, 7868. [CrossRef]
16. Almomani, O. A Feature Selection Model for Network Intrusion Detection System Based on PSO, GWO, FFA and GA Algorithms. *Symmetry* **2020**, *12*, 1046. [CrossRef]
17. Majeed, P.G.; Kumar, S. Genetic algorithms in intrusion detection systems: A survey. *Int. J. Innov. Appl. Stud.* **2014**, *5*, 233.
18. Dhaliwal, S.S.; Nahid, A.A.; Abbas, R. Effective Intrusion Detection System Using XGBoost. *Information* **2018**, *9*, 149. [CrossRef]

19. Chen, Z.; Jiang, F.; Cheng, Y.; Gu, X.; Liu, W.; Peng, J. XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud. In Proceedings of the 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, China, 15–17 January 2018; pp. 251–256. [\[CrossRef\]](#)
20. Louk, M.H.L.; Tama, B.A. Dual-IDS: A bagging-based gradient boosting decision tree model for network anomaly intrusion detection system. *Expert Syst. Appl.* **2023**, *213*, 119030. [\[CrossRef\]](#)
21. Kabla, A.H.H.; Thamrin, A.H.; Anbar, M.; Manickam, S.; Karuppayah, S. PeerAmbush: Multi-Layer Perceptron to Detect Peer-to-Peer Botnet. *Symmetry* **2022**, *14*, 2483. [\[CrossRef\]](#)
22. Wang, L.; Cheng, Z.; Lv, Q.; Wang, Y.; Zhang, S.; Huang, W. ACG: Attack Classification on Encrypted Network Traffic using Graph Convolution Attention Networks. In Proceedings of the 2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Rio de Janeiro, Brazil, 24–26 May 2023; pp. 47–52. [\[CrossRef\]](#)
23. Fernandes, R.; Lopes, N. Network Intrusion Detection Packet Classification with the HIKARI-2021 Dataset: A study on ML Algorithms. In Proceedings of the 2022 10th International Symposium on Digital Forensics and Security (ISDFS), İstanbul, Turkey, 6–7 June 2022; pp. 1–5. [\[CrossRef\]](#)
24. Umar, M.A.; Chen, Z.; Shuaib, K.; Liu, Y. Effects of Feature Selection and Normalization on Network Intrusion Detection. *TechRxiv* **2024**. [\[CrossRef\]](#)
25. Rai, A. Optimizing a New Intrusion Detection System Using Ensemble Methods and Deep Neural Network. In Proceedings of the 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 15–17 June 2020; pp. 527–532. [\[CrossRef\]](#)
26. Desale, K.S.; Ade, R. Genetic algorithm based feature selection approach for effective intrusion detection system. In Proceedings of the 2015 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 8–10 January 2015; pp. 1–6. [\[CrossRef\]](#)
27. de Almeida Florencio, F.; Moreno, E.D.; Teixeira Macedo, H.; de Britto Salgueiro, R.J.P.; Barreto do Nascimento, F.; Oliveira Santos, F.A. Intrusion Detection via MLP Neural Network Using an Arduino Embedded System. In Proceedings of the 2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC), Salvador, Brazil, 5–8 November 2018; pp. 190–195. [\[CrossRef\]](#)
28. Idrissi, I.; Azizi, M.M.; Moussaoui, O. A Lightweight Optimized Deep Learning-based Host-Intrusion Detection System Deployed on the Edge for IoT. *Int. J. Comput. Digit. Syst.* **2021**, *11*. [\[CrossRef\]](#) [\[PubMed\]](#)
29. Sahlan, F.; Feizal, F.Z.; Mansor, H. Home Intruder Detection System using Machine Learning and IoT. *Int. J. Perceptive Cogn. Comput.* **2022**, *8*, 56–60.
30. Fernandes, R.; Silva, J.; Ribeiro, O.; Portela, I.; Lopes, N. The impact of identifiable features in ML Classification algorithms with the HIKARI-2021 Dataset. In Proceedings of the 2023 11th International Symposium on Digital Forensics and Security (ISDFS), Chattanooga, TN, USA, 11–12 May 2023; pp. 1–5. [\[CrossRef\]](#)
31. Abdulhammed, R.; Faezipour, M.; Musafer, H.; Abuzneid, A. Efficient Network Intrusion Detection Using PCA-Based Dimensionality Reduction of Features. In Proceedings of the 2019 International Symposium on Networks, Computers and Communications (ISNCC), İstanbul, Turkey, 18–20 June 2019; pp. 1–6. [\[CrossRef\]](#)
32. Heba, F.E.; Darwish, A.; Hassanien, A.E.; Abraham, A. Principle components analysis and Support Vector Machine based Intrusion Detection System. In Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, Cairo, Egypt, 29 November–1 December 2010; pp. 363–367. [\[CrossRef\]](#)
33. Subba, B.; Biswas, S.; Karmakar, S. Intrusion Detection Systems using Linear Discriminant Analysis and Logistic Regression. In Proceedings of the 2015 Annual IEEE India Conference (INDICON), New Delhi, India, 17–20 December 2015; pp. 1–6. [\[CrossRef\]](#)
34. Sumaiya Thaseen, I.; Aswani Kumar, C. Intrusion detection model using fusion of chi-square feature selection and multi class SVM. *J. King Saud Univ. Comput. Inf. Sci.* **2017**, *29*, 462–472. [\[CrossRef\]](#)
35. Mirjalili, S.; Mirjalili, S. Genetic algorithm. In *Evolutionary Algorithms and Neural Networks: Theory and Applications*; Springer: Cham, Switzerland, 2019; pp. 43–55.
36. Jebbari, K.; Madiafi, M. Selection methods for genetic algorithms. *Int. J. Emerg. Sci.* **2013**, *3*, 333–344.
37. Pachuaui, J.L.; Roy, A.; Kumar Saha, A. An overview of crossover techniques in genetic algorithm. In *Modeling, Simulation and Optimization: Proceedings of CoMSO 2020*; Springer: Singapore, 2021; pp. 581–598.
38. XGBoost. XGBoost Classifier Parameters. Available online: <https://xgboost.readthedocs.io/en/stable/parameter.html> (accessed on 2 January 2024).
39. NVIDIA. Jetson Nano Developer Kit. Available online: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit> (accessed on 21 December 2023).
40. Tharwat, A.; Gaber, T.; Ibrahim, A.; Hassanien, A.E. Linear discriminant analysis: A detailed tutorial. *AI Commun.* **2017**, *30*, 169–190. [\[CrossRef\]](#)
41. Zaman, S.; Karray, F. Collaborative architecture for distributed intrusion detection system. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009; pp. 1–7. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.