Autonomous Network Intrusion Detection for Resource-Constrained Devices of the Internet of Things

Jefferson Cavalcante[13], Tiago G. F. Barros[12], Jose N de Souza[3]

[1]Centro de Estudos e Sistemas Avançados do Recife (CESAR)

[2]CESAR School

[3]Universidade Federal do Ceará (UFC)

jrac@cesar.org.br, tgfb@cesar.school, neuman@ufc.br

Abstract. *With the Internet of Things (IoT), cars, home assistants, cameras and other devices may be part of an ubiquitous network with access to personal information in real-time, able to interact with the environment and even influence people. For this reason, empowering such devices with network intrusion detection algorithms might be vital for the development of a more secure and trustworthy Internet of Things. In this work we study the performance and accuracy of machine learning models trained for this task, and analyzed the impact of their deployment in a microcontroller-based System-on-Chip used by IoT devices, along with the impact of TinyML-based optimizations, such as vectorization and quantization. From our experiments, Decision Trees presented very low inference time of 5 microseconds on average, and higher accuracy of 99.9% when compared to Logistic Regression and Neural Networks, being a viable solution for real-time, accurate and autonomous Network Intrusion Detection system for IoT devices.*

1. Introduction

Computationally efficient techniques and purpose-specific hardware allow the processing of artificial intelligence algorithms right at the edge [Abadade et al. 2023]. Executing such algorithms as close as possible to where data is generated and collected, avoids transmission of potentially sensitive information to the cloud. However, if end-devices gather, process and act without sending data to edge or cloud servers, they might be the only device with access to the environment and thus become the primary target for cyber attackers.

With the Internet of Things (IoT), cars, home assistants, cameras and other devices may be part of an ubiquitous network with access to personal information in real-time, able to interact with the environment and even influence people. For this reason, securing such devices against cyberattacks is vital for the development of a more secure and trustworthy Internet of Things.

Recent works point to the use of Machine Learning (ML) for Network Intrusion Detection (NID), however most of these works do not consider their deployment on IoT devices [Aldhaheri et al. 2024], relying on gateways, edge or cloud servers to run the NID system, as depicted in Figure-1(a) and Figure-1(b). Due to the nature of the Internet of Things, with devices either directly or indirectly connected to the Internet, we believe

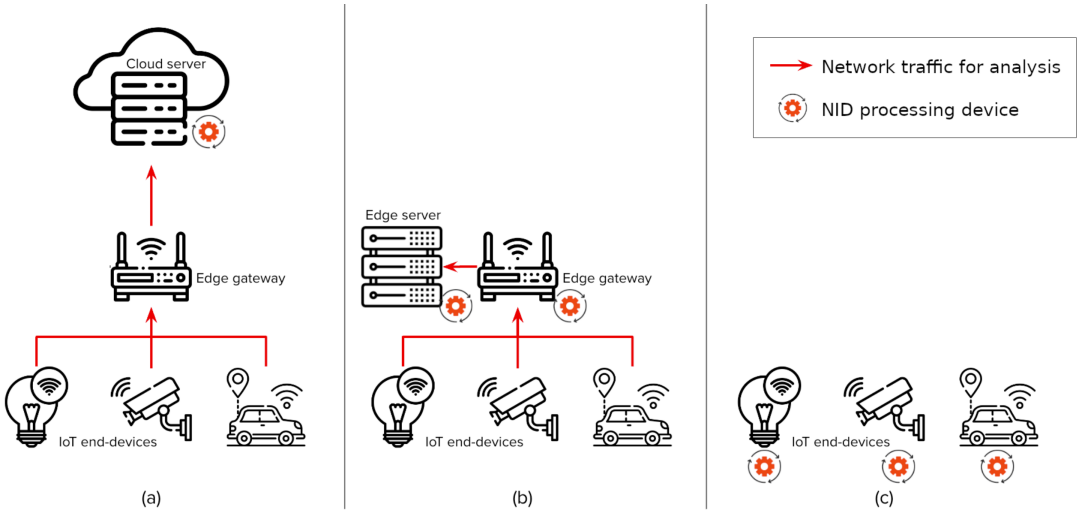

Figure 1. Deployments of Network Intrusion Detection systems: (a) on cloud servers, (b) on edge servers or gateways, (c) on end-devices. Source: Elaborated by the authors, using images from Flaticon.com.

On-Device Network Intrusion Detection (OD-NID) should be part of their processing, as depicted in Figure-1(c), instead of relying on the existence of other devices analysing network traffic [Rahman et al. 2020], which increases network complexity and might not even be present in all networks.

Current machine learning techniques for embedded systems [Abadade et al. 2023] enable On-Device Network Intrusion Detection. However since computationally constrained devices, called Low Power Low Connectivity (LPLC) Devices by ITU-T[ITU-T 2019], might suffer from low memory footprint and low processing power, and NID is a side task for them, deployment of NID solutions on IoT devices should consume as few resources as possible to be feasible and accessible for a broad range of devices. Therefore, the contributions of this work are:

- Evaluate accuracy, average inference duration and memory requirements of different machine learning models performing Network Intrusion Detection on a real IoT device;
- Evaluate the impact of quantization and vectorization on accuracy, average inference duration and memory requirements of the models under evaluation;
- Based on experiments, provide insights on the deployment of machine learning models in real-world IoT devices for On-Device Network Intrusion Detection.

This work is organized as follows: Section 2 introduces recent works on Network Intrusion Detection for the Internet of Things, Section 3 presents approaches for efficient execution of machine-learning-based NID on IoT end-devices, Section 4 discusses experiments to assess performance and accuracy of Network Intrusion Detection on such devices, Section 5 presents the results and Section 6 discuss final conclusions and future works.

2. Related works

Machine learning techniques for NID have been gaining importance in the last decade. Many surveys have been carried out to investigate ML-based approaches for Intru-

sion Detection Systems (IDS) in general [Aldhaheri et al. 2024, Chaabouni et al. 2019, Da Costa et al. 2019, Ferrag et al. 2020, Hajiheidari et al. 2019].

T. Chua and I. Salam[Chua and Salam 2023] evaluated ML algorithms for NID using 3 datasets for Network Intrusion Detection in a progressive strategy, with most recent data from the dataset being separated for tests. The idea is that the testing dataset can contain some changes in the attack type that reflects how the attacks evolve across the time, and in this way, can assess the long-term performance of the algorithms. The experiments did not involve execution on IoT devices.

P. F Araujo-Filho et. al [Freitas De Araujo-Filho et al. 2021] presented an Intrusion Prevention System (IPS) for automotive controller area network (CAN). The proposed detection mechanism uses iForest model to detect malicious frames at a high rate (78μs) and fires a discard frame command before their transmission is completed. The experiments were executed in a Raspberry Pi 4B.

The work in [Tekin et al. 2023] evaluates energy consumption of ML models trained for network intrusion detection running on cloud and edge servers, a single-board computer (Raspberry Pi 4B) and a microcontroller-based IoT device. However, it doesn't analyze the impact of vectorization and quantization for the deployment of models on IoT devices.

Work in [Rahman et al. 2020] proposes a Federated Learnning (FL) approach for intrusion detection in IoT. In this approach, a generic model is sent to an end-device, improved using local data and then, the model parameters are sent back to a server that aggregates the weights and improves the model. This approach executes the ML algorithms on end-devices in the learning phase.

Evaluating the set of works presented in this section, that use different approaches for intrusion detection applied to different fields, it is possible to realize that the vast majority of their experiments do not take into account the execution of ML models on LPLC devices, since most of them are executed in laptops or in a Raspberry Pi 4B, that has between 2GB and 8GB of RAM and a 64-bit quad-core processor @ 1.8GHz, comparable to a desktop computer.

Thus, the need to embed NID in LPLC devices is latent and contributes to a major advance towards safer IoT devices and networks.

3. On-Device Network Intrusion Detection

Recent works on Network Intrusion Detection present a current trend in the use of machine learning models due to their high accuracy, adaptability to various datasets and potential for compute-time optimizations [Aldhaheri et al. 2024] [Ferrag et al. 2020]. Deploying ML models into microcontroller-based IoT devices requires minimization of computational resources usage while keeping high accuracy, since this is a side task for IoT end-devices but privacy is at risk when intrusion detection fails. In this section we introduce AI models used for Network Intrusion Detection in the context of IoT devices, and introduce TinyML techniques that can reduce memory and computational overhead under certain conditions, and can be suitable for the deployment of ML-based NID solutions on end-devices in the Internet of Things.

3.1. Machine Learning approaches

From recent works on NID, we listed some machine learning models for classification that are commonly evaluated, such as Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), Artificial Neural Networks (ANN), Support Vector Machines (SVM) and K-Nearest Neighbors (KNN) [Chua and Salam 2023, Tekin et al. 2023, Da Costa et al. 2019]. However, some of them might not suit well for inference in LPLC devices: KNN which requires the device to store the train dataset for inference; RF and SVM usually present highest inference times. For this reason, we chose Logistic Regression, Decision Tree and MultiLayer Perceptron (MLP), a type of ANN, to analyze, since they are powerful and present important differences: LR performs a dot product between the input and its weights vector followed by a non-linear activation function; MLP expands that to multiple neurons organized in layers, each performing a dot product followed by a non-linear activation; and DT leverage branch instructions to perform comparisons and take decisions.

Both Logistic Regression and MultiLayer Perceptron rely on vector operations of floating point numbers, which can be optimized using Single Input Multiple Data (SIMD) instructions when available. Models trained with floating point numbers can also be converted to use fixed-point arithmetic via quantization, which uses integer instead of floating point arithmetic, potentially reducing storage and execution time.

3.2. TinyML optimization techniques

Artificial Intelligence (AI) has been used in a wide range of areas, such as healthcare, autonomous driving, natural language processing and more, but usually require expensive specialized hardware for faster training and inference, sometimes with the use of cloud computing for cost reduction, which imposes constraints and brings concerns regarding data privacy, the need for constant connectivity and high carbon footprint.

TinyML emerged with various approaches to reduce memory and processing requirements for the computations involved in processing AI, either during training or inference, commonly relying on hardware-specific capabilities for more efficient and accurate execution [Abadade et al. 2023]. Two approaches are broadly used for the deployment of AI models on microcontroller-based devices: quantization and vectorization.

3.2.1. Quantization

is used to convert floating-point into fixed-point arithmetic, which for large models dominate processing time due to matrix-vector multiplications performed by various machine learning models [Vanhoucke et al. 2011]. It also reduces memory requirements, since floating point numbers are usually stored in 32-bit or 16-bit formats, but when in fixed-point they are usually stored in 8-bit integers or smaller.

Quantization imposes a trade-off between model size and accuracy, since it introduces quantization error the smaller the integers used [Abadade et al. 2023]. Also, quantizing inputs and intermediate vectors require floating point arithmetic between floats and integers, which are usually not hardware-accelerated, thus might not be a viable approach if model size is small enough to not pay-off those conversions.

3.2.2. Vectorization

relies on hardware-specific instructions, the so-called Single Instruction Multiple Data (SIMD), that perform multiple operations with a single instruction. It aceelerates multiplication and addition of multiple pairs of numbers for example, very useful for various machine learning models, like neural networks, which require potentially massive amounts of matrix-vector multiplications. In the past, the introduction of Intel's Streaming SIMD Extensions (SSE) and AMD's 3DNow! paved the way for AI algorithms running on consumer Personal Computers [Vanhoucke et al. 2011]. Nowadays, microcontrollers with support for SIMD instructions have achieved unprecedent performance processing AI algorithms [Abadade et al. 2023], giving rise to the area of Artificial Intelligence of Things (AIoT).

3.3. Deployment strategy

As illustrated in Figure-1, NID systems are usually deployed in the cloud or at the edge, but on a range of devices.

Figure-1(a) illustrates their deployment in the cloud, where cloud servers are used due to high availability of compute resources and centralized setup, alleviating the configuration overhead performed in IoT networks. However, this type of deployment requires that information about network traffic is sent to the cloud in real-time for further processing, creating communication overhead in the network, requiring constant connectivity to the cloud and introducing delays.

Figure-1(b) illustrates deployment of NID systems at the edge, where processing occur on edge servers, gateways or both. This setup requires that every IoT network have its own dedicated hardware setup for the processing of network traffic information, which can make such deployment infeasible in various scenarios where users lack the knowledge to setup such systems and networks. This setup also suffers from the need for collection of traffic information, which can add communication overhead to the network.

Finally, Figure-1(c) illustrates On-Device Network Intrusion Detection, in which detection is performed by each and every device in the network on their own behalf, alleviating the need for any setup in the network for this purpose, which has the potential to broaden adoption of intrusion detection. Also, no communication overhead is added to the network for the purpose of disseminating traffic information. With the use of lightweight algorithms and TinyML techniques, the impact of performing On-Device Network Intrusion Detection can be minimized, easing adoption by different vendors on a wide range of types of IoT devices.

From an on-device perspective, Figure-2 shows the packet processing steps introduced by an AI-based on-device Network Intrusion Detection System (NIDS). What information and statistics from each particular flow will be computed depends on the AI model feature set. A larger corpus of features might incur in higher processing overhead. Besides that, converting flow information into an input vector for the NIDS AI model is another source of additional overhead, and varies from model to model. For example, if data normalization or quantization is required, it may involve costly computations. For this reason, care must be taken when choosing an AI model to avoid introduce too much overhead in this step.

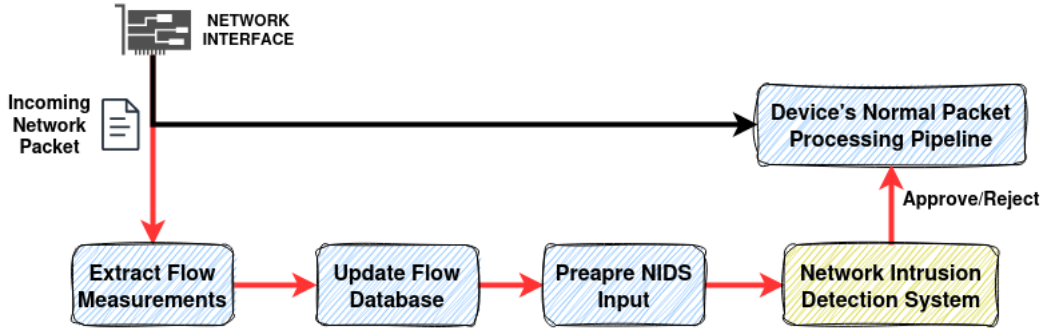

Figure 2. On-device packet processing pipeline. Red lines indicate NIDS-specific processing steps.

4. Experiments

To investigate how Machine Learning models trained for Network Intrusion Detection perform on embedded devices, we trained 3 machine learning models on a dataset, then deployed multiple versions of them, each with a different TinyML-based optimization technique. During experiments we measured their accuracy and average inference time, which in turn affect throughput, since each network packet has to be analyzed when in production.

4.1. Dataset

Supervised models require a dataset with ground truth information for their learning phase. For that purpose, we chose the LUFlow, a recent dataset which contains large amounts of samples from real-world traffic and was built to serve as a strong ground truth [Mills et al. 2022]. The LUFlow dataset contains flow-based information from traffic collected in a honeypot setup inside of the Lancaster University. It autonomously classifies traffic as benign when comes from services inside their own network, malign when targeted at the honeypot, and anomalous when traffic did not fit within the typical monitoring profile and is kept for further analysis.

For data pre-processing and features selection, we followed the procedure from [Chua and Salam 2023], which analyzes features importance and recommends the use of a subset of them. The importance of features selection in this scenario is beyond the high dimensionality problem. In LPLC devices, more features means more operations which might increase inference time. By applying the procedures described in [Chua and Salam 2023], we chose the subset of features described in Table-1.

4.2. Methodology

System-on-a-chip (SoC) microcontrollers arised as cheap yet powerful platoforms for expediting the development of Internet of Things applications. They usually embed support for wireless communications via WiFi and Bluetooth, integrate with cloud services like Microsoft's Azure IoT and Amazon's AWS IoT and provide hardware-accelerated vector operations which enables the use of TinyML techniques like vectorization and quantization. The ESP32S3 SoC, shown in figure 3, is an important microcontroller of this class and was chosen for use in our experiments.

Table 1. LUFlow features selected

Features	Descriptions
bytes_in	Cumulative number of bytes received
bytes_out	Cumulative number of bytes sent
dest_port	Flow's receiver port number
entropy	Flow's data entropy in bits per byte
num_pkts_out	Cumulative number of packets sent
num_pkts_in	Cumulative number of packets received
proto	Protocol number
src_port	Sender's port number
total_entropy	Entropy from all data fields of the flow in bytes
avg_ipt	Average of the flow's inter-packet transmission time

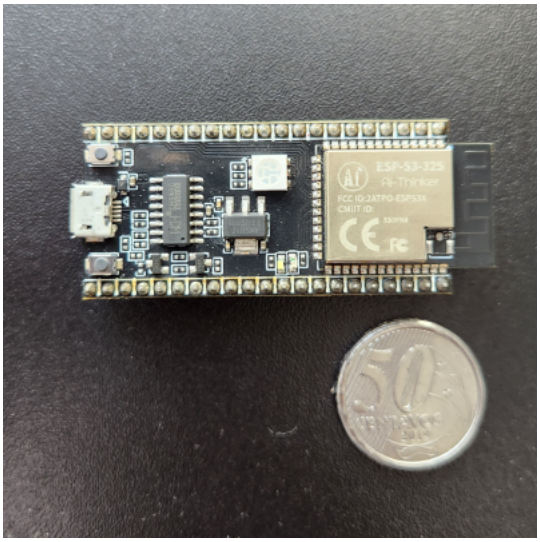

Figure 3. IoT device with an ESP32S3 chip.

To evaluate the performance of machine learning models on IoT devices and the impact of vectorization and quantization to memory usage, accuracy and inference time, we embedded variations of Logistic Regression, MultiLayer Perceptron and Decision Tree classifiers into an ESP32S3-based device. They were trained with data from 20 days of monitoring of The LUFlow dataset, a total of 12603010 samples from fev/2021 and june/2022 with equal number of malign and benign traffic classes. The dataset was split into train and test sets, with 10082408 and 2520602 samples respectively, but only the first 10000 samples from the test set were used for evaluation. During the experiments, 10000 samples were statically embedded into the device's flash memory and a message via its serial port indicates which model should be used to process all of them, then all outputs along with total inference time were sent back via the serial port.

For the training, Logistic Regression and Decision Tree models were implemented with the SK-Learn python framework, and MultiLayer Perceptrons were implemented with the Pytorch framework. To reduce the need for computationally expensive non-linear activation kernels, the ReLU activation was used in the MLP hidden layer, that can be implemented with an if condition inside of a loop. Also, we experimented with hidden

layers of 3 sizes: 16, 32 and 64 neurons. However, accuracy didn't differ much, while inference time and size increased noticeably, so for the purposes of comparison with other models, we chose to only use the 16-neuron MLP in the experiments.

All models were also implemented in C to be compiled to native code for the ESP32S3 chip. Model parameters were copied from the python implementation and stored as static arrays. Each model was implemented with floating and fixed-point arithmetic, and some were also implemented with support for vectorization using hardware-specific instructions supported by the ESP32S3 chip. All implementations are described in Table-2.

Table 2. Implementations of each machine learning model.

Model implementation	Description
LR	Floating point LR
LR SIMD	Floating point LR with vectorization
LR INT8	Fixed-point LR
LR INT8 SIMD	Fixed-point LR with vectorization
MLP	Floating point MLP
MLP SIMD	Floating point MLP with vectorization
MLP INT8	Fixed-point MLP
MLP INT8 SIMD	Fixed-point MLP with vectorization
DT	Floating point DT
DT INT8	Fixed-point DT

Model quantization without the use of SIMD was implemented in C using 8-bit integers for weights quantized in the Q.7 fixed point format. When using SIMD, we leveraged a library called IDF-DSP from the vendor of ESP32S3 chips, which requires 16-bit integers in the case of quantized models, we stored Q.7 numbers in 16-bit integers and a slight difference in precision can be observed due to different implementation details. For the decision tree, quantization was performed using the Q.15 fixed point format, since from experiments we noticed that Decision Trees are more sensitive to the quantization scheme than Neural Networks or Logistic Regression.

During experiments, three metrics are measured: average inference time, calculated by dividing the total execution time by the number of test samples; accuracy, computed as the number of correct classifications divided by the total number of test samples; and memory required to store model parameters. It is important to note that input quantization is included in the total duration used to compute the average inference time.

5. Results

This section presents results from the execution of all model implementations for Network Intrusion Detection in an ESP32S3 SoC.

Figure-4 presents the average duration of an inference. The first noticeable result is that Logistic Regression and Decision Trees spent up to 5 microseconds per inference on average, which translates into the processing of more than 200000 packets per second. It is also noticeable that quantized models had significantly higher inference times, that's because since the models are already very small, quantizing the input means performing

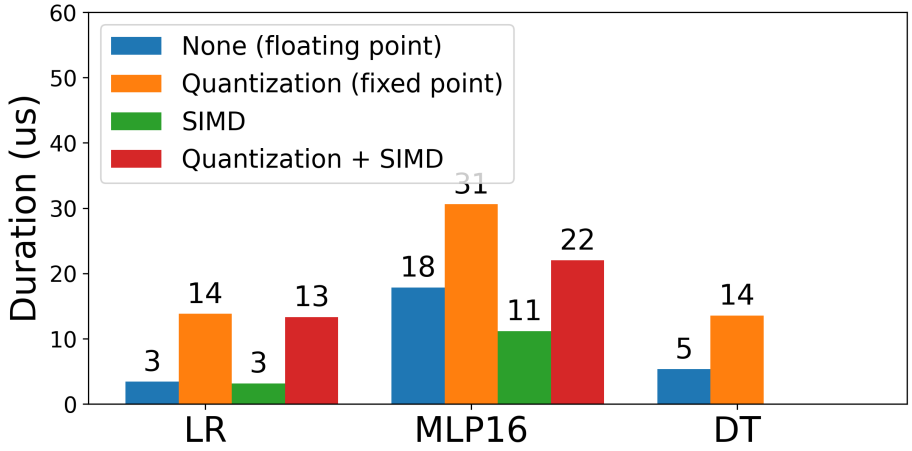

Figure 4. Average inference time.

floating-point arithmetic on the input vector, which accounts for a significant portion of the computing time. With MLPs, quantization is also performed after each hidden layer. For such small models in a device with a floating point unit, quantization actually might no be recommended, which is an important result. When comparing the impact of SIMD, it is noticeable that it had an important impact on MLPs, which need to perform more vector operations. From these results, Decision Trees and Logistic Regression were the best models for deployment so that minimal impact on performance is caused.

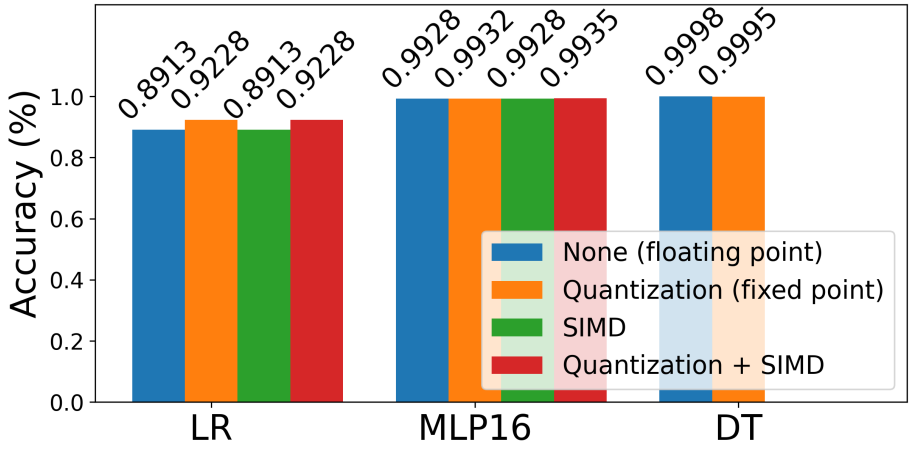

Figure 5. Accuracy.

Figure-5 presents the accuracy of each model after processing 10000 test samples. From these results, Logistic Regression had worse accuracy to the point that it might not be suited for real-world deployments. Decision Trees and MultiLayer Perceptrons on the other hand achieved more than 99% of accuracy on all implementations, which are very impressive results given their model size and inference time. From all models implementations, the floating point Decision Tree presented highest accuracy.

It is important to observe how each model classifies benign and malicious traffic individually, so we guarantee they don´t overspecialize in only one class. Figure-7 presents confusion matrices of models with and without quantization. Both MLPs and

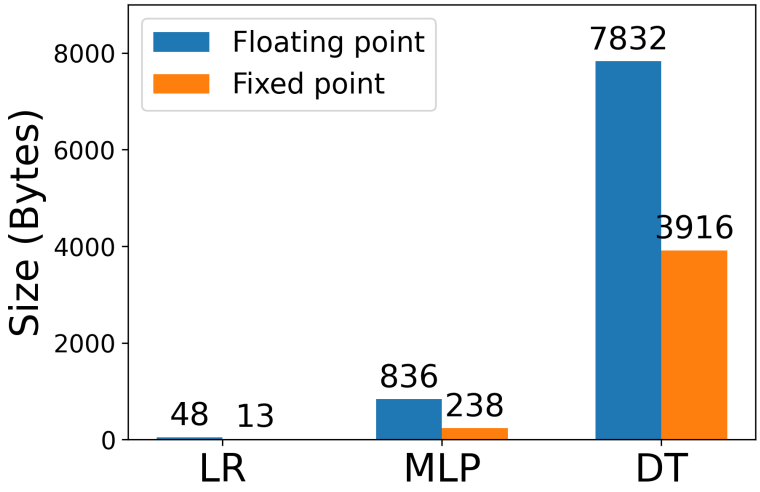

Figure 6. Model sizes based on the number of parameters.

DTs were slightly better at classifying negative cases when compared to positive, but overall all models did very well in dealing with both classes.

Finally, one important aspect of deploying ML models on IoT end-devices is the amount of storage required. Figure-6 compares how much memory each morel requires to store its parameters. From that analysis, Decision Trees required more memory then the Logistic Regression and Neural Networks models, which must be taken into consideration if storage is a strict constraint.

As a lesson learned from the experiments presented in this work, with such small models, inference is fast enough to make input quantization and normalization a potencial performance bottleneck, since they can involve floating point arithmetic during conversions, as can be seen in Figure-4. However, some machine learning models such as decision trees are insensitive to data normalization, and most of the selected features are already integers. Thus, for future research, the combination of models insensitive to feature scaling and the choice for integer-only features could be explored for lower inference latency without involving a Floating-Point Unit (FPU) when turning flow information into input features for the AI model of the on-device NIDS.

6. Conclusions

Standalone Network Intrusion Detection algorithms can become an important role for the future IoT devices deployed at houses, offices, streets and other places where privacy is of utmost importance. This work studied the deployment of machine learning models trained for this specific task on an IoT end-device, and analyzed the impact of quantization and vectorization on memory, accuracy and inference time, three important measures which translates into device's cost, throughput, energy consumption and response time.

From our results, Decision Trees were the best models for this task, presenting higher accuracy when compared to Neural Networks and Logistic Regression, with highest accuracy and lowest inference times when compared to the same competitors, although at the cost of more memory for the storage of its parameters. We also noticed that quantization seems to have reduced model overfitting and slightly increased accuracy but at the

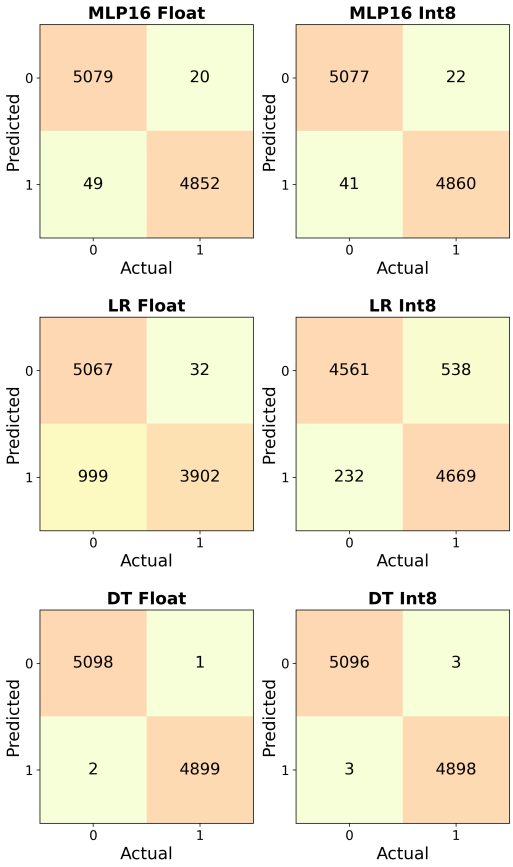

Figure 7. Confusion matrix.

cost of much higher inference time, since all models were very small and matrix multiplications accelerated by quantization did not have as much impact as floating point operations involved in quantizing the input and intermediate vectors. Finally, the availability of SIMD instructions in the device reduced inference time for MLPs, which perform various matrix-vector multiplications, and can be very useful in reducing latency with such models.

For recent future work, we plan to combine multiple datasets to improve model reliability, implement more machine learning models, and deploy them on more IoT devices with chips of different architectures, providing more insights to help accelerate the adoption of Network Intrusion Detection in microcontroller-based connected devices for a more secure and trustworthy Internet of Things.

References

Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., and Hafid, A. S. (2023). A comprehensive survey on tinyml. *IEEE Access*, 11:96892–96922.

Aldhaheri, A., Alwahedi, F., Ferrag, M. A., and Battah, A. (2024). Deep learning for cyber threat detection in iot networks: A review. *Internet of Things and Cyber-Physical Systems*, 4:110–128.

Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., and Faruki, P. (2019). Network intrusion detection for iot security based on learning techniques. *IEEE Communications Surveys & Tutorials*, 21(3):2671–2701.

Chua, T.-H. and Salam, I. (2023). Evaluation of machine learning algorithms in network-based intrusion detection using progressive dataset. *Symmetry*, 15(6).

Da Costa, K. A., Papa, J. P., Lisboa, C. O., Munoz, R., and de Albuquerque, V. H. C. (2019). Internet of things: A survey on machine learning-based intrusion detection approaches. *Computer Networks*, 151:147–157.

Ferrag, M. A., Maglaras, L., Moschoyiannis, S., and Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50:102419.

Freitas De Araujo-Filho, P., Pinheiro, A. J., Kaddoum, G., Campelo, D. R., and Soares, F. L. (2021). An efficient intrusion prevention system for can: Hindering cyber-attacks with a low-cost platform. *IEEE Access*, 9:166855–166869.

Hajiheidari, S., Wakil, K., Badri, M., and Navimipour, N. J. (2019). Intrusion detection systems in the internet of things: A comprehensive investigation. *Computer Networks*, 160:165–191.

ITU-T (2019). Y.4460 - architectural reference models of devices for internet of things applications.

Mills, R., Marnerides, A. K., Broadbent, M., and Race, N. (2022). Practical intrusion detection of emerging threats. *IEEE Transactions on Network and Service Management*, 19(1):582–600.

Rahman, S. A., Tout, H., Talhi, C., and Mourad, A. (2020). Internet of things intrusion detection: Centralized, on-device, or federated learning? *IEEE Network*, 34(6):310–317.

Tekin, N., Acar, A., Aris, A., Uluagac, A. S., and Gungor, V. C. (2023). Energy consumption of on-device machine learning models for iot intrusion detection. *Internet of Things*, 21:100670.

Vanhoucke, V., Senior, A., and Mao, M. Z. (2011). Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*.