Optimizing Intrusion Detection for SDN-Enabled IoT Using of Lightweight Deep Learning Architectures

Mustafa T. Saleh[1] and Ali H. Hamad[2(✉)]

[1] Informatics Institute for Postgraduate Studies, University of Information and Communication Technology, Baghdad, Iraq
ahamad@kecbu.uobaghdad.edu.iq
[2] Biomedical Application Department, College of Artificial Intelligence, University of Baghdad, Baghdad, Iraq
phd202230706@iips.edu.iq

Abstract. The rising use of the Internet of Things (IoT) has led to the use of Software-Defined Networking (SDN) as a scalable framework that organizes large amounts of heterogeneous data traffic. However, the centralization nature of SDN present sturdy security gaps, with they being distributed denial-of-service (DDoS) attacks to advanced intrusion vectors, which threaten the privacy of governmental and trade confidential information. Although Deep Learning (DL) represents a disruptive step towards the optimization of a network, as well as remediation, the sheer volume of computation often conflicts with the constrained capabilities of edge gateways and IoT endpoints. This work poses the question of the effectiveness of four canonical DL architectures, including deep neural networks (DNNs), recurrent neural networks (RNNs), convolutional neural networks (CNNs), and long short-term memory units (LSTMs) in an SDN-based intrusion detection system. The exploration, based on the SICA dataset, which contains more than 2.2 million incident records of 22 different typologies of attacks, follows three major model-compressing approaches, namely structural pruning, weight clustering, and quantization. The empirical results indicate that even though LSTM provides the best classification accuracy, over 95$\mathscr{R}$, a pruned DNN provides a good trade off in the deployment of edges with a ten times processing throughput with insignificant loss in detection confidence. Such findings provide a stringent blueprint that organizations aiming to adopt real-time, resource-saving cybersecurity defenses in the current digital ecosystem may use.

Keywords: IDS · IoT · Deep learning · Pruning · QAT · Clustering

1 Introduction

As the Internet of Things (IoT) grows at a fast rate, scores of trillions of interconnected devices are producing enormous amounts of data whilst it is demanding efficient, reliable, and adaptive communication strategies. IoT is a heterogeneous entity networks,

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16759, pp. 641–656, 2027.
https://doi.org/10.1007/978-3-032-30518-3_40

bandwidth constraints, and latency sensitivity, and there are significant difficulties with energy constraints of equipment to traditional paradigms of communication. The use of algorithms has been done to boost efficiency in communication in solving optimization issues. However, traditional optimization models are not always in line with the dynamic, data-oriented aspect of the Internet of Things systems. As a result, deep learning proposes new optimization procedures and takes part in their refined development. Smart learning mechanisms facilitate real time reactions and decision making that are data based. Deep optimization through learning has the potential of being transformative. IoT communication is limited by coordination of network resources, coordination of devices and intelligent provision of data. Thus, the systematic research on how deep learning methods could be applied to creating more efficient optimization algorithms of the Internet of Things is urgently required [1]. The protection of sensitive information, intellectual property, government, and trade secrets are important in the modern digital environment. Cybersecurity issues become increasingly challenging due to the technical progress. Despite some significant interest in cybersecurity education and training, researchers continue to struggle with effectively surveying different approaches. This research intends to critically evaluate frameworks and ways to develop and establish new cybersecurity training solutions to individuals and organizations [2].

SDN has become a big step in the evolution of the area of network architecture, a new era of efficiency, flexibility and programmability. At the heart of SDN is the separation of the control plane and the data plane, which is a groundbreaking idea that enables network immediate power of administrators over traffic in the network flow and management [3]. Nor can this paradigm shift, though conducive also, optimizations and agile network configurations to illuminate a range of security risks present in the SDN architecture. The SDN control is centralized coupled with the programmability and openness of plane. SDN environments are especially made by network interfaces prone to various online attacks, including Denial of Service (DDoS) attacks to advanced intrusion attempts. Considering such challenges, there is need of strong, dynamic, and proactive security is more eminent than ever. Among the bombard of the security solutions, it is IDS an essential line of defense, which is intended to identify, evaluate and control the bad practices in the network. However, the peculiarities of SDN demand a re-consideration and redesign of conventional IDS strategies in order to suitably manage the vulnerabilities peculiar to SDN infrastructures. The further tightens cyber threats, which are becoming more sophisticated this necessity, which demands novel solutions which can not only identify known attack vectors, but also predict and eliminate emerging threats [4].

2 Related Work

This part presents the latest developments in Intrusion Detection Systems (IDS) and draws particular attention to such methods as quantization, pruning, and optimization to allow the use of them on resource-restricted Internet of Things (IoT) and edge computing devices.

Idrissi, et al. (2022) [5] discussed the possibility of implementing Deep Learning-oriented Host-Intrusion Detection System (DL-HIDS) on commercial IoT devices. The authors used various optimizations that were specific to specific hardware to achieve

constrained memory and inference timing constraints. Luo (2023) [6] suggested a distributed SDN-based IDS, which makes use of a decision tree that is optimized according to the Black Hole Optimization (BHO) algorithm. Pruning of the model is done with the help of BHO to maximize the accuracy of each sub-network. Sharmila and Nagapadma (2023) [7] introduced Quantized Autoencoder (QAE) which is an unsupervised model that includes pruning, clustering, and integer quantization (uint8 and float16 versions in particular). Their model of QAE-u8 has a reduction of 92.23% in memory occupancy with high detection accuracy. Wang et al. (2023) [8] introduced a lightweight model named DL-BiLSTM, which is based on Deep Neural Networks (DNN) and Bidirectional Long Short-Term Memory (BiLSTM). It employs Incremental Principal Component Analysis (IPCA) to minimize dimensionality and dynamic quantization so as to minimize computation overheads on internet of things devices.

Acharya, et al. (2024) [9] suggested a design architecture that is based on Quantization-aware Neural Architectural Search to evolve quantized neural network models automatically. Such models are much smaller than the state-of-the-art networks and are designed with the FPGAs in mind. Huynh and Nguyen (2024) [10] proposed a better three-step training model of edge computing devices. It uses model compression and a better pruning techniqueology to compress model sizes yet maintain high accuracy on the UNSW-NB15 dataset. Bouayad, et al. (2024) [11] proposed Lightweight-Fed-NIDS that is a federated learning system with structured model pruning. One of the contributions is the single time calculation of a pruning mask to optimize local models without client access information, and pruning size is cut by 90%. Morea and Kachavimatha (2025) [12] suggested the method of Software-Defined Networking (SDN)-based intrusion detection based on meta-heuristic feature selection combined with a K-Nearest Neighbors (KNN) classifier to detect malicious actions in real-time. Rangsikunpum, et al. (2025) [13] concerned with in-vehicle networks, and the article introduces an ultra-lightweight IDS implemented with the use of Binarised Neural Networks (BNN). The authors reduced the number of parameters by 91.07% through network pruning to get a loss (accuracy) of 0.1% (Table 1).

Table 1. Comparative of previous work

Ref.	Methodology	Dataset(s)	Optimization Techniques	Key Results
Idrissi, et al. (2022)	Deep Learning-based Host-IDS (DL-HIDS)	Specific commercial IoT traffic	Hardware-specific optimizations for edge deployment	Optimized for memory consumption and inference timing on commercial IoT devices.

(*continued*)

Table 1. (*continued*)

Ref.	Methodology	Dataset(s)	Optimization Techniques	Key Results
Sharmila Nagapadma (2023)	Quantized Autoencoder (QAE)	RT-IoT2022 (Real-Time IoT 2022)	Pruning, clustering, and integer quantization (uint8 and float16)	QAE-u8 achieved a 92.23% reduction in memory size and 70.01% lower average memory utilization.
Luo (2023)	Distributed SDN-based IDS using optimized forests (Decision Trees)	NSL-KDD and UNSW-NB15	Black Hole Optimization (BHO) for tree pruning and node splitting	Achieved 99.2% accuracy on NSL-KDD and 97.2% on UNSW-NB15.
Wang, et al. (2023)	DL-BiLSTM: Combined DNN and Bidirectional LSTM	CIC-IDS2017, N-BaIoT, and CICIoT2023	IPCA for dimensionality reduction; Dynamic Quantization for cell structure	Surpassed traditional models in performance while maintaining lower complexity and model size.
Acharya, et al. (2024)	Quantization-aware Neural Architectural Search (NAS)	Not explicitly stated in abstract snippet	Automated training and evolution of quantized neural networks	Models are 1,000x smaller than state-of-the-art; 2.3x to 8.5x fewer LUTs when deployed on FPGAs.
Huynh & Nguyen (2024)	Enhanced Three-Stage Training (DSD) paradigm	UNSW-NB15	Enhanced pruning methodology and model compression	Significantly reduced model dimensions while maintaining accuracy levels of larger models.
Bouayad et al. (2024)	Lightweight-Fed-NIDS: Federated Learning with CNNs (ResNet, VGG)	UNSW-NB15, USTC-TFC2016, and CIC-IDS-2017	Structured model pruning via a one-time pruning mask	90% reduction in model size; 3X acceleration in training time; ~99% detection rate.

(*continued*)

Table 1. (*continued*)

Ref.	Methodology	Dataset(s)	Optimization Techniques	Key Results
Morea and Kachavimatha (2025)	SDN IDS using meta-heuristic feature selection	Not explicitly stated in abstract snippet	Meta-heuristic optimization for feature selection	Enabled real-time analytics and improved classification for SDN vulnerabilities.
Rangsikunpum, et al. (2025)	Pruned Binarised Neural Networks (BNN) for In-Vehicle Networks	CH and SA datasets	Binarization (1-bit weights) and network pruning	91.07% reduction in parameters with only 0.1% accuracy loss; optimized for FPGA/Jetson Orin Nano.

3 Methodology

Designing a lightweight Intrusion Detection System (IDS) for the Internet of Things (IoT) requires a methodology that balances high-security precision with the harsh hardware constraints of edge devices. Unlike cloud-based systems, an IoT-ready IDS must minimize its memory footprint and computational latency to operate on microcontrollers or gateways. The following methodology outlines the systematic transition from a complex Deep Learning (DL) model to a lightweight, deployable security solution.

3.1 Intrusion Detection System

The Intrusion Detection Systems (IDS) are complex security systems that continuously scan network traffic and system activities to identify uncharacteristic patterns or familiar threats, and alert administrators when they are detected. These can be categorized in two main dimensions namely the method used to detect and the extent of monitoring. This is based on the detection methodology into two broad categories of IDS. The former is Signature Detection (also known as Misuse based Intrusion Detection). This approach is based on a continuously maintained list of signatures, or patterns or fingerprints that identify a known attack and suspicious behavior. The IDS scans network packets or system events and compares it against this repository of archived signatures; a hit generates an alert and as a result, identifies a recognized threat. The second is the Behavioral Anomaly Detection which is a method that creates a benchmark of normal network or system behavior. Any difference with this set normal base is perceived as an anomaly and a possible intrusion. To create a dynamic profile of normal activities over time, a statistical model, machine learning algorithm or other methods are used, and the real-time activity is compared against it [14].

IDS is also bifurcated when the classification is done based on the scope of monitoring. First, the Network-Based Intrusion Detection Systems (NIDS) monitor the traffic of a network segment and analyze it in real-time to detect any threats that pass through the network. Placed at strategic points in the network, such as at the perimeter or major routing points such as switches and routers, NIDS will use network taps or switch mirror ports (SPAN) to obtain all packet data including headers and payloads on the entire watched portion of the network. Secondly, Host-Based Intrusion Detection Systems (HIDS) are software agents that are run on a specific host (servers, laptops, workstations) and thematically on that specific machine monitors activity. The agent of a HIDS is directly embedded to the operating system of every host machine [15].

3.2 Light Weight Techniques

Typically, edge devices cannot perform refined computations, making it critical to evolve and apply more efficient, lightweight models suited to their limitations. Model pruning, quantization, and clustering are the most efficient optimization methodologies that are exploited to make deep transfer learning models more lightweight [16]. These methodologies work on optimizing both the models' size and performance by eliminating any unnecessary parameters and making the computations needed during inference much easier. Decreasing the models' size represents one of the many benefits of lightweight schemes, as well as extended accessibility, improved privacy levels, and reduced costs. However, there is a reduction in the overall speed of the model. There are also downsides, like complexity, limited flexibility, low accuracy, and trade-offs between models' accuracy, speed, and size. There are three main deep learning model light weight techniques, in the following a brief description for each:

(i) Pruning

Pruning is a type of optimization method that can be used to make a model smaller eliminating irrelevant connections, neurons or parameters. Connection pruning consist of eliminating weights with values equal to or almost equal to zero in an attempt to reduce the memory needs of the whole network. The model that is pruned can be re-trained and in fact have better accuracy than the original model. Figure 1 shows a basic sample of a pre-pruned model i.e. with all neurons connected and the result simplified model with less connections. One of the most widespread pruning techniques is magnitude-based pruning and has attained significant performance with respect to decreasing memory demands of machine learning models without sacrificing the accuracy of the scheme. This technique consists of removing weights with values that are below a specified threshold of a trained model. Pruning may also be adopted by specifying the pruning ratio (PR) of the model which specifies the weight percentages to be zeroed in a whole network or the individual layers to be trimmed by the use of a pruning schedule [17]. The pruning process could be summarized as follows:

$$w'_{ij} \begin{cases} w_{ij} \ \ if \ \ w_{ij} \geq \theta_0 \\ 0 \ \ if \ \ w_{ij} < \theta_0 \end{cases} \tag{1}$$

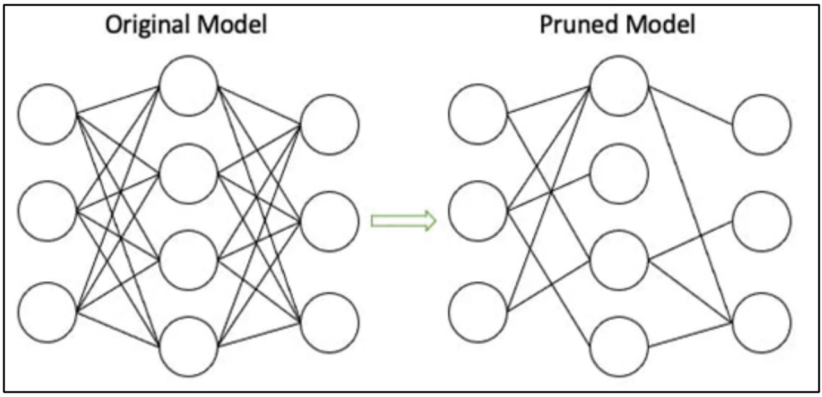


Fig. 1. Pruning light weight technique [17].

(ii) Quantization

Quantization is a model optimization method, the goal of which is to reducing the size of a DL model by quantizing weight values to a reduced precision as bits, that normally is between 32 bits and 16 bits. Moreover, it can be possible to represent binary floating-point numbers [18]. Quantization is separable into two major classes post-training quantization (PTQ) without retrain the model. Secondly, quantization-aware training (QAT) which required retrain the model. Figure 2 indicates the FP32, FP16, FP8 and INT8 value representations in a computer system. It is noted that various representations make use of varying bits count. The precision is associated with the size of bits on which a value is represented. The closer to reality the depiction is, the larger the size of the memory required to store the model. This process is executed by the application of the quantization function q, which is defined in a point wise manner.

$$quantize_{range} = \left[2^{(num\ bit-1)}, 2^{(num-1)} - 1\right] \tag{2}$$

$$scale\ factor\ \ weight = \frac{\max(weight)}{2^{(num_{bit}-1)}} \tag{3}$$

$$quantized\ value = round\left(\frac{value}{scale_{factor}}\right) \tag{4}$$

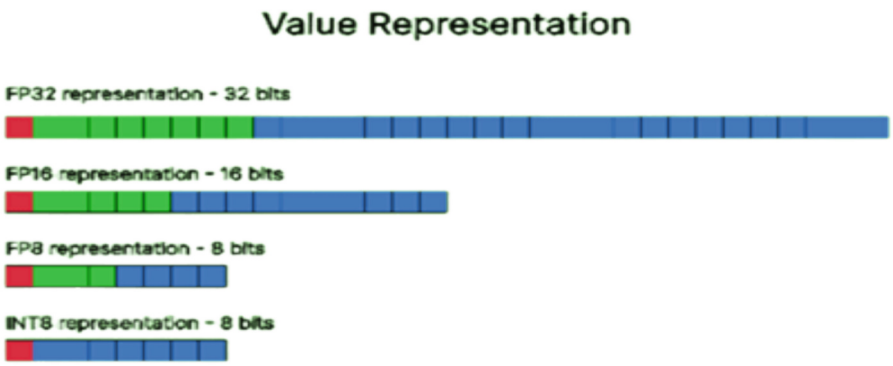


Fig. 2. Quantization principle [18].

(iii) Weight Clustering

Weight clustering or weight sharing is an optimization method which is composed of finding a representative value of a cluster or a group of weights, thereby reducing the amount of different possible values that would be represented in the network. It means that many different. The neurons within a network have the same weight. An example of simple weight clustering is shown in figure 3 that clustered sixteen weights in four categories identified by a color. Four reference values are associated with this case the average of the values of each cluster, and the weight of each is also quantized to one of the reference values mentioned above. The green cluster as an example contains five elements with different values. The average of these components is 0.15 which is the reference value of this cluster. Having used weight sharing, everything in green has a weight of 0.15 weight value. In the case of this particular cluster, the parameters are substituted by a single one. An integer that is the reference index and FP32 value. In the example, taking all the clusters into account, the network with weight clustering has instead of sixteen different FP32 values substitutes these special values with four FP32 values, therefore, optimizing the memory storage [19].

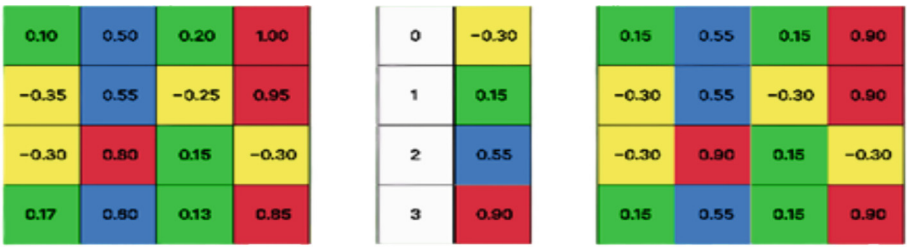


Fig. 3. Weight clustering technique [19]

4 Proposed System Design

To achieve an empirically sound and holistic dataset, we are compelled to incorporate a wide range of application services on the experimental corpus. This approach allows to simulate subtly attack vectors that are contemporary and Internet-based and are embedded in a software-defined networks (SDN) setting. Besides, the attack scenarios should include more advanced paths which will cross several SDN elements. Available academic sources cover a significant range of such situations, both in-house and external with regards to SDN ecosystem. To capture this, we outline four sub-domains we are going to deploy to Mininet operating Ubuntu 20.04 LTS, hence establishing concrete topology. These sub-domains are linked through two Ryu controllers and two switching the sub-domains through four OpenFlow vSwitch (OVS) switches. A Unix-like open-source Python controller, the Ryu framework stands out by being able to support the challenging needs of modern networks infrastructures and remain scalable at the same time. Its wide interoperability with divergent protocols of SDN and functional capabilities make it useful in disparate network designs, as well as it is susceptible to heterogeneous integration. The first two sub-domains simulate a classic network stack, which serves a broad array of services including HTTP and FTP servers. On the other hand, the other two sub-domains simulate an Internet of Things (IoT) network specifically intended to be interrogated in the course of experimentation, both indoors and outdoors. The structure of the proposed multi-controller SDN is supported in figure 4.

In this work, four deep learning models, i.e. DNN, RNN, CNN and LSTM were used. These models are determining the suggested intrusion detection system (IDS) and performance light weighting methods benchmarking. The hyperparameters of the suggested algorithms are listed in Table 2. After the training of each DL model, it was pruned, quantized, and clustered to become light weighted. The proposed algorithms were tested in terms of model sizes before and after light weighting, other training and testing measurements like accuracy and loss. An open-source software-defined networking (SDN) controller, called RYU controller, along with Mininet, a network emulator, were used to create an SDN network environment, thus a realistic SDN architecture was emulated. The simulated SDN created standard traffic as well as attack traffic of different types. It can be assumed that Metasploit and hping3 might have been used to model a wide variety of attack vectors such as exploits, denial-of-service attacks, etc., but ping utility was used to send normal traffic. An unprocessed network traffic data including various attack and normal traffic was captured by a packet analyzer like Wireshark. To create a single pcap (Packet Capture), a ubiquitous format to save network packets, all of the separate traffic captures were amalgamated into a single pcap. The statistical analysis of the suggested implementation of SICA dataset is available in the Table 3.

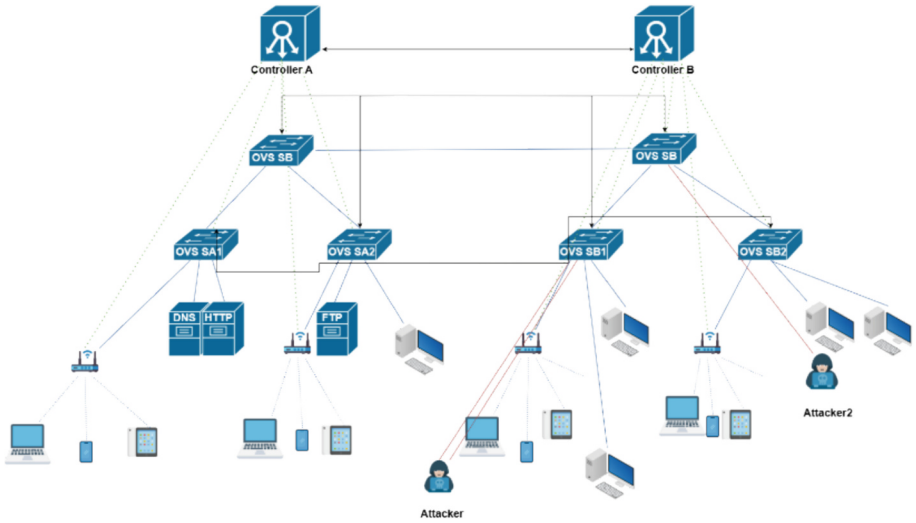


Fig. 4. Proposed multi-controller SDN

Table 2. Deep learning models hyperparmeters

Model	Layers	Neurons	Hyperparameters
LSTM	Input → LSTM (5x) → Output	256 → 128 → 128 → 64 → 32 → Output	Activations: Tanh (hidden), Softmax (output).
CNN	Input → Conv1D → MaxPool → Conv1D → MaxPool → Conv1D → MaxPool → Flatten → Dense (2x) → Output	256 → 128 → 128 → 64 → 32 → Output	Activations: ReLU (hidden), Softmax (output), Kernal size = 2,3.
RNN	Input → GRU → Dense (2x) → Output	64 → 64 → 32 → Output	Activations: ReLU (hidden), Softmax (output).
DNN	Input → Dense (5x) → Output	256 → 128 → 128 → 64 → 32 → Output	Activations: ReLU (hidden), Softmax (output).

Table 3. Statistical analysis of proposed SICA

Parameters	Description
Dataset	SICA
No. of Records	2,298,903
No. of Features	84
No. of attack classes	22
Dataset size	1262 MB
attack types	'Arp_spoofing' 'apache_range_dos' 'arp_sweep' 'avahi_portzero' 'brute_dirs' 'ddos' 'fragmentation_attack' 'http_flood' 'http_get_uri_long' 'http_get_uri_strings' 'normal' 'ntp_prot' 'ntp_protocol_fuzzer' 'slowloris_attacks' 'smb2_negotiate_corrupt' 'smb_tree_connect_corrupt' 'ssh' 'ssh_version_corrupt' 'synflood' 'teardrop' 'udp_probe' 'udp_reflection'

5 Results and Discussion

Based on the results shown in figure 5 a,b, and c from the SICA dataset across the three optimization techniques, RNN models were seen as close contenders but the LSTM had over 95% accuracy throughout the board but the RNN was seen to be much more efficient in terms of model size since it was much lighter (about 10.8 KB) than the DNN and LSTM (about 2627 KB). Conversely, the 1D-CNN showed the worst performance, with the accuracies of about 71–72% which means that it would not be suitable in the current dataset. By and large, despite some trends to use quantization as a method of hardware acceleration, pruning and clustering were found to be equally effective when it comes to saving the accuracy of the larger DNN and LSTM models, though the RNN is the best solution where extreme memory limitations are involved due to its relatively small footprint.

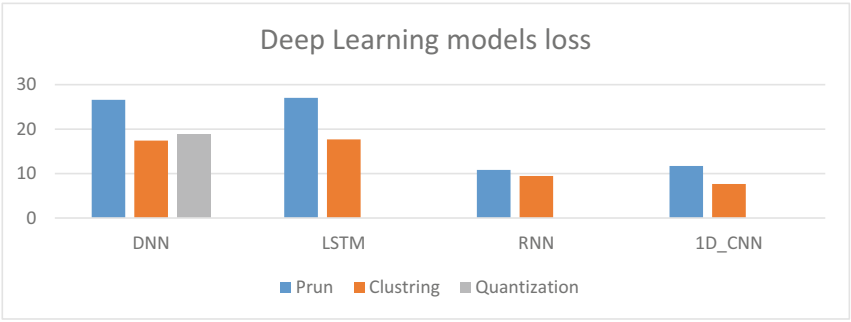
The training process based on quantization-awareness (QAT) generated numerical values only on the DNN model, which did not converge when using LSTM, RNN, and 1DCNN. Despite the results of the DNN, its accuracy dropped to 83.6% (QAT) as compared to 96.5% (pruned). This is an indication that the DNN was stable enough to be trained, but quantization was a big performance cost. Since LSTMs and RNNs are more sensitive than DNNs, they must have reached a zero utility or technical error at some stage of training. A number of technical reasons give rise to the common failure of QAT to deliver results - or get circumvented - on recurrent and convolutional architectures compared to plain dense networks:

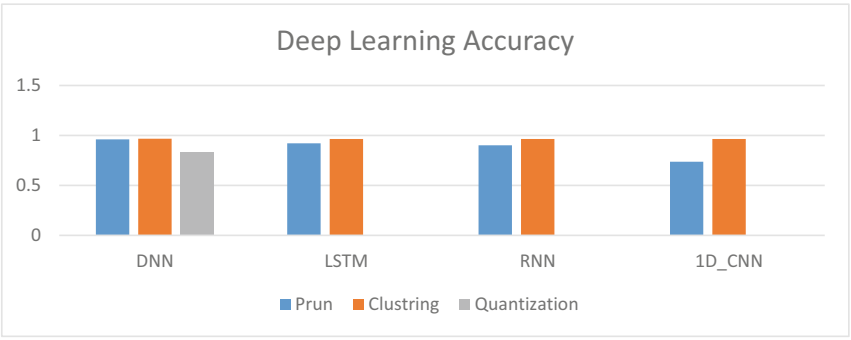
- Recurrent Networks (RNN and LSTM): These networks are sequential and process data at a time, with an output of one time step serving as an input of the next stage. Fake quantization nodes are customarily inserted in QAT to simulate low-precision arithmetic. The microscopic error of the quantization process becomes magnified at every time step, often leading to gradient instability or exploding/vanishing gradients, and the training process will fail or the model will simply blow up.

- Dense Networks (DNNs): Dense feed-forward networks have a generally smaller number of depth per sample than the unrolled time-step sequences of a LSTM. They are therefore much more resistant to the added noise caused at the quantization-aware training pipeline.
- RNN/CNN Layer Complexity: Quantization of LSTM or 1DCNN layers is more complex since to quantize them, one must quantize several internal elements at once: gates, cell states, kernels and biases. In case the particularisation of these layers in the one's code does not fit in the QAT wrapper, the training will either crash, or yield invalid results (e.g., NaN).

Table 4 shows a comparative study of four canonical deep leaning frameworks (LSTM, RNN, DNN, and CNN) against their beheaded, "lightweight" versions. Through the implementation of structural pruning, weight clustering and quantization aware training (QAT) the investigation aims to determine an equilibrium between the confidence of classification and computational expediency. The empirical results highlight a strong trade-off: while optimizations greatly reduce the latency, they often introduce a phenomenon we call "label drift," where specific instances of the attack will be re-labeled into more generic categories. Baseline performance for both the LSTM and RNN architectures are quite good, especially on detection of DDoS and BFA attacks. Nonetheless, these temporally-aware models are excessively susceptible to clustering procedures. When clustering is in force both models lose the ability to distinguish BFA from a probe attacks and instead classify them as DDoS. This suggests that the shared weight enforced in clustering disrupts the fine gate mechanisms relevant to memory essential for achieving sequence-learning based inference. Moreover, while pruning is typically followed by the goal of speeding up the inference process, the LSTM/RNN mod results show a small amount of latency, from 0.06 ms to around 0.09 ms, which is often the result of failing to optimally exploit sparse matrix operations by the execution platform.

The DNN (Deep Neural Network) appears as the most "compression-friendliest" topology among the various deep network topologies in this study. Diverging from the recurrent counterparts, the DNN maintains high confidence levels involving each optimization paradigm applied. Of particular note, the values of both pruned and clustered versions of the DNN exhibit the smallest latency values shown in the whole table, down to a whopping 0.00475 ms. Like this CNN has the benefit of pruning as well but the serious decline of confidence when clustering (with the scores collapsing into the 0.35–0.41 range). This observation suggests that although CNNs can be significantly accelerated, clustering convolutional kernels in CNNs may remove spatial features that are needed for confident intrusion detection. A recurring theme throughout all experimental conditions is the difficulty of proper classification of the probe attacks. Even in baseline models, the confidence involved in probing is significantly worse than in DDoS detection. DNN using QAT the model produces agile and fine-grained labels like [*http flood*] and [*apacherange_dos*]. This pattern suggests that quantification could be an attempt to regularize the learner or that it is QAT itself that allows for the capture of more granular attack subclasses. On the other hand, most variants that are not heavy tend to resort to a "DDoS" classification in the absence of knowledge, which is a sign of loss of feature depth.





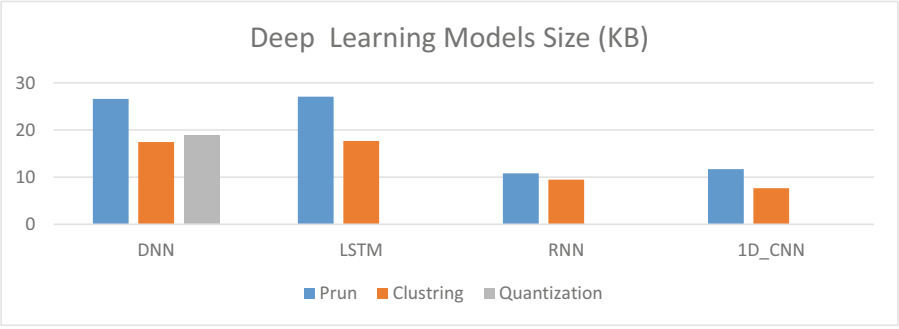


Fig. 5. Deep learning model test results. a) Loss, b) Accuracy, c) Model size

Table 4. Classification testing of real-time traffic on each dataset for 300 packets

Attack Category	Classified Label	Confidence	Time (ms)	Classified Label	Confidence	Time (ms)	Classified Label	Confidence	Time (ms)	Classified Label	Confidence	Time (ms)
Baseline model				LSTM Prun			LSTM **Cluster**			LSTM QAT		
BFA	BFA	0.8219	0.06	DDoS	0.798335	0.0925	DDoS	0.719197	0.097	-	-	-
DDoS	DDoS	0.9656	0.06	DDoS	0.78747	0.09325	DDoS	0.86546	0.10365	-	-	-
Probe	Probe	0.2956	0.26	Probe	0.41147	0.087018	DDoS	0.491566	0.093158	-	-	-
Baseline model				**RNN Prun**			**RNN Cluster**			**RNN QAT**		
BFA	BFA	0.7303	0.06	DDoS	0.626135	0.0705	DDoS	0.905968	0.1505	-	-	-
DDoS	DDoS	0.9669	0.06	DDoS	0.707992	0.07605	DDoS	0.905091	0.11185	-	-	-
Probe	Probe	0.3414	0.25	Probe	0.546567	0.072281	DDoS	0.623057	0.107193	-	-	-
Baseline model				**DNN Prun**			**DNN Cluster**			**DNN QAT**		
BFA	BFA	0.8212	0.06	DDoS	0.850372	0.005	BFA	0.849724	0.0055	[http_flood]	0.715332	0.00775
DDoS	DDoS	0.9802	0.06	DDoS	0.807679	0.0071	DDoS	0.821406	0.00475	[http_flood]	0.796465	0.00845
Probe	udp_reflection	0.7002	0.31	DDoS	0.528366	0.005439	Probe	0.703944	0.004912	[apache_range_dos]	0.37685	0.007895
Baseline model				**CNN Prune**			**CNN Cluster**			**Baseline model**		
BFA	BFA	0.8432	0.06	DDoS	0.380788	0.0115	BFA	0.750968	0.0105	-	-	-
DDoS	DDoS	0.9545	0.07	DDoS	0.379386	0.0148	DDoS	0.774739	0.0104	-	-	-
Probe	BFA	0.7728	0.24	DDoS	0.351902	0.010877	DDoS	0.416424	0.01	-	-	-

In sum, if the operating goal is maximum inference speed for edge deployed intrusion detection applications, the pruned DNN is the best option and it offers a speedup of 10x with virtually no tradeoff in confidence. On the other hand, when the most important question is the exact detection of complex attack patterns, then the baseline LSTM is still better because the interventions to make it a lightweight model (specifically clustering for consistency) adversely affect its predictive acuity. The evidence gathered here suggests that, for security critical deployments, pruning is the better optimization strategy and will always maintain the integrity of the original classification schema.

6 Conclusion

The present study fulfilled the dire need of systematic research on efficient optimization algorithms for IoT security. We used a realistic SDN test bed based on Mininet and Ryu controllers to demonstrate that standard IDS methods have to be reconfigured to manage the special vulnerabilities of the SDN architectures. Our relative assessment of light weighting measures showed that there was an apparent tradeoff between the efficiency of computation and fidelity to classification. Pruning proved to be the most stable procedure, it maintained the original schema of classification but greatly diminished the size of the models. DNN architectures were very good at compression with latencies as low as 0.00475 milliseconds. Recurrent models (LSTM and RNN) were found to be highly accurate; nevertheless, they were vulnerable to clustering and quantization components, resulting in such phenomena as label drift or training collapses due to unstable gradients. In applications where maximum inference speed is the prime consideration, pruned DNN is the best solution. On the other hand, in cases where there is a high need to detect complex patterns of attacks with a high level of accuracy, due to the involvement of protection of sensitive data and government secrets, the normal LSTM would provide better performance, regardless of the higher computational requirements. The focus of future research should be on optimization of sparse matrix operations in order to reduce the gap between the complexity of the model and performance of the edge- devices in real-time.

References

1. Guo, L., Li, X., Si, W.: Research on deep learning-based optimization algorithms for IoT communication. IEEE Access. **13**, 177724–177724 (2025). https://doi.org/10.1109/ACCESS.2025.3597098
2. Di Gennaro, F., Cucchiarelli, A., Morbidoni, C., Spalazzi, L.: A comparative analysis of datasets for intrusion detection in software-defined networks. In: Joint National Conference on Cybersecurity (ITASEC & SERICS 2025) (2025)
3. Khorseed, W.S., Hamad, A.H.: Inter and intra domain DDOS attack mitigation for software defined network based on hyperledger fabric blockchain technology. Ingénierie Des Systèmes D Inf. **29**(1), 301–311 (2024). https://doi.org/10.18280/isi.290130
4. Janabi, A. H., Kanakis, T., Mark Johnson: Survey: intrusion detection system in software-defined networking. In Barbara Guidi (Ed.), IEEE Access **12**, 164097 (2024). https://doi.org/10.1109/ACCESS.2024.3493384

5. Idrissi, I., Moussaoui, O., Azizi, M.: A lightweight optimized deep learning-based host-intrusion detection system deployed on the edge for IoT. Int. J. Comput. Digit. Syst. **11**(1), 209–216 (2022). https://doi.org/10.12785/ijcds/110117
6. Luo, K.: A distributed SDN-based intrusion detection system for IoT using optimized forests. PLoS One. **18**(8), e0290694 (2023). https://doi.org/10.1371/journal.pone.0290694
7. Sharmila, B.S., Nagapadma, R.: Quantized autoencoder (QAE) intrusion detection system for anomaly detection in resource-constrained IoT devices using RT-IoT2022 dataset. Cybersecurity. **6**(1), 41 (2023). https://doi.org/10.1186/s42400-023-00178-5
8. Wang, Z., Chen, H., Yang, S., Luo, X., Li, D., Wang, J.: A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization. PeerJ Comput. Sci. **9**, e1569 (2023). https://doi.org/10.7717/peerj-cs.1569
9. Yu, R., Le Jeune, L., Mentens, N.: University of Florida, KU Leuven, & Worcester Polytechnic Institute. Quantization-aware neural architectural search for intrusion detection. arXiv. https://arxiv.org/pdf/2311.04194 (2024)
10. Huynh, T.T., Nguyen, H.T.: Effective multi-stage training model for edge computing devices in intrusion detection. Int. J. Comput. Netw. Commun. **16**(1), 13–31 (2024). https://doi.org/10.5121/ijcnc.2024.16102
11. Bouayad, A., Alami, H., Janati Idrissi, M., Berrada, I.: Lightweight federated learning for efficient network intrusion detection. In S. D. Campbell (Ed.), Access **12**, 172027 (2024). https://doi.org/10.1109/ACCESS.2024.3494057
12. More, S.A., Kachavimath, A.V.: SDN intrusion detection using meta-heuristic optimization and K-nearest Neighbors classifier. Procedia Comput. Sci. **260**, 1137–1144 (2025). https://doi.org/10.1016/j.procs.2025.03.299
13. Rangsikunpum, A., Amiri, S., Ost, L.: Ultra-lightweight and highly efficient pruned binarised neural networks for intrusion detection in in-vehicle networks. Electronics. **14**(9), 1710 (2025). https://doi.org/10.3390/electronics14091710
14. Dhirar, H., Hamad, A.: Internet of things software-defined network intrusion detection dataset: inter- and intra-domain. Al-Khwarizmi Eng. J. **21**(3), 35–47 (2025). https://doi.org/10.22153/kej.2025.08.001
15. Dhirar, H., Hamad, A.H.: Federated deep learning intrusion detection system on software defined-network based internet of things. IAES Int. J. Artif. Intell. (IJ-AI). **14**(4), 3109–3120 (2025) http://ijai.iaescore.com
16. Ferreira, G., Da Nóbrega Marinho, M.H., Severo, V., Madeiro, F.: Optimization strategies applied to deep learning models for image steganalysis: application of pruning, quantization and weight clustering. Appl. Sci. **15**(9), 4632 (2025). https://doi.org/10.3390/app15094632
17. Han, S., , Mao, H., , Dally, W. J.: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In ICLR 2016. https://arxiv.org/pdf/1510.00149 (2016)
18. Jacob, B., et al.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2704–2713, Salt Lake City, UT, USA (2018). https://10.1109/CVPR.2018.00286
19. Zhang, M., Ma, Y., Li, J., Zhang, J.: A density connection weight-based clustering approach for dataset with density-sparse region. Expert Syst. Appl. **230**, 120633 (2023). https://doi.org/10.1016/j.eswa.2023.120633