

Collaborative Lightweight Deep Learning Optimization for Intrusion Detection in SDN-IoT Networks

Ali H. Hamad^{*1}, Mustafa T. Saleh²

¹Department of Biomedical Application, College of Artificial Intelligence, University of Baghdad, Baghdad, Iraq

²Informatics Institute for Postgraduate Studies, University of Information and Communications Technology, Baghdad, Iraq

Correspondence

* Ali Hussein Hamad

Department of Biomedical Application, College of Artificial Intelligence, University of Baghdad, Baghdad, Iraq

Email: ahamad@kecbu.uobaghdad.edu.iq

Abstract

The combination of the Internet of Things (IoT) and Software-Defined Networking (SDN) has quickly revolutionized the world of industry, and also brought with it some severe security risks. Despite its recent success, Deep learning (DL) can be a powerful method for Network Intrusion Detection systems (NIDS), but often the computational and memory requirements are too high for edge devices. In this paper, a collaborative optimization framework with lightweight deep-learning IDS models suitable for SDN-IoT environments is proposed. We formalize and test three collaborative compression schemes based on combining pruning, clustering, and quantization, yet maintaining the structural advantages of each stage: Sparsity-Preserving Quantization-Aware Training (PQAT), Cluster-Preserving Quantization-Aware Training (CQAT), and Sparsity-and-Cluster-Preserving Quantization-Aware Training (PCQAT). The three schemes are evaluated on four canonical architectures (LSTM, RNN, DNN and CNN) on the proposed SICA dataset with respect to DDoS, Probe and Brute-Force (BFA). The results show that aggressive single-stage compression such as clustering can shrink model size by up to 94%, but at the cost of predictive confidence; PCQAT, by contrast, acts as a stabilizer that recovers this lost confidence. In particular, CNN-PCQAT and DNN-PCQAT deliver the most favorable trade-off, sustaining high detection accuracy on all three attack classes with inference latency below 0.01ms. The framework therefore offers a practical, deployment-oriented guideline for implementing real-time IDS on resource-constrained SDN-IoT edge devices.

Keywords

Software Defined Network (SDN), Intrusion Detection System (IDS), IoT, Pruning, Clustering, Quantization, PQAT, CQAT, PCQAT.

I. INTRODUCTION

The IoT technology is gaining pace in the latest century. Due to its ability to facilitate autonomous data transfer with human intervention, it can be used in an extremely large variety of applications in a wide field of industries and disciplines, such as agriculture, unmanned vehicles, healthcare, and manufacturing. It is expected that the IoT will transform the way humans exist because it will create more autonomous and wholly intelligent systems. However, the very existence of the current IoT (Internet of Things) device environment as a publicly deployed environment gives them a vulnerability to both cyber and physical attacks. Security issues, as to user-privacy violations caused by computer attack, may be particularly harmful. Therefore, there is an urgent need to come up with a powerful and dynamic intrusion-detection system that can deal with the security issues that come with the IoT [1].

Combining Software-Defined Networking (SDN) with the Internet of Things (IoT) is radically changing modern network architectures by increasing flexibility, scalability,

and centralized control. While SDN is able to separate the data plane from the control plane, enabling programmable and dynamic network customization, IoT is capable of connecting billions of diverse devices, creating vast amounts of data. Yet, the integration brings in new and complex security challenges, most of which are attributed to the distributed, resource-constrained and heterogeneous characteristics of IoT ecosystems, as well as the centralized control logic of SDN. Intrusion Detection Systems (IDS) are therefore important elements towards the protection of SDN-IoT networks. They scan traffic to detect an activity that is malicious and breach of policies. Traditional IDS solutions can be unsuccessful when they have to face the dynamic, heterogeneous nature of SDN-IoT worlds. The Deep Learning (DL) techniques are powerful ones which can automatically learn complex patterns from vast amounts of data. The effectiveness of DL-based IDS systems, however, highly relies on the quality, relevance, and format of the data sets for training and testing [2].

To facilitate IDS research on a scale where large volumes of publicly available datasets can be employed, research



This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.
© 2026 The Authors.

datasets have been constructed such as BoT-IoT [3], ToN-IoT [4], InSDN [5] and SDN-IoT [6] which deal with various aspects of network behavior and attack typologies. Such datasets might not sufficiently represent complexity of modern SDN IoT architectures, or may be biased in the representation of the different categories of attacks. To tackle this problem, the present paper introduces a novel, synthetically generated Network Based-Intrusion Detection Systems (NIDS) dataset especially designed to simulate various attack scenarios along with realistic distributions of traffic in SDN-IoT networks. The comparative analysis of the new IDS dataset SDN-IoT with InSDN, SDN-IoT, BoT-IoT and ToN-IoT validates its efficiency.

IDS solutions have long been used as a security measure in conventional computer networks in order to fulfill the diversified security needs. However, the traditional IDS would not be very suitable in IoT networks due to a number of corresponding limitations, such as the poor computational and storage capabilities of the devices of the IoT. The general trend has been to conduct numerous studies that take into account security preconditions and resource constraints of IoT devices, also known as lightweight IDS. These solutions however, are normally associated with low Detection Rate (DR), high False Negative (FN) rate and high False Positive (FP) rate. In addition, most of such studies do not evaluate their models based on CPU or memory consumption [7].

Despite this progress, three interconnected gaps still hinder the practical deployment of deep-learning-based IDS on SDN-IoT infrastructures. First, existing benchmark datasets such as InSDN, BoT-IoT, ToN-IoT, and SDN-IoT inadequately capture the traffic heterogeneity, device diversity, and attack breadth that characterize contemporary SDN-IoT deployments, which constrains the generalizability of the resulting models. Second, conventional compression techniques-pruning, clustering, and quantization, each impose a stark trade-off between footprint reduction and detection fidelity when applied in isolation; furthermore, naively chaining them often causes one stage to nullify the structural benefits of the previous stage. Clustering, for instance, destroys the sparsity introduced by pruning, while post-training quantization can erase the centroid structure produced by clustering. Third, the literature lacks a systematic, head-to-head comparison of collaborative compression strategies across canonical deep-learning architectures (LSTM, RNN, DNN, and CNN) under realistic SDN-IoT attack scenarios, reported jointly against the deployment-critical metrics of model size, inference latency, and confidence. These gaps motivate the present work, which introduces a unified collaborative optimization framework that preserves the structural benefits of each compression stage and delivers deployable, real-time intrusion detection on resource-constrained edge devices.

In this work, a collaborative light weight deep learning models have been proposed which are Sparsity preserving quantization aware training (PQAT), Cluster preserving quantization aware training (CQAT), and Sparsity and cluster preserving quantization aware training (PCQAT). The dilemma that arises when one attempts to put these methodologies together is the fact that the results of any one of the techniques traditionally annihilates the results of the precedent method and thus one must undermine the overall benefit of using all of them at the same time; to give an example, clustering does not preserve the sparsity induced by the pruning API.

The main contributions of this paper are summarized as follows:

- We construct and release SICA (SDN-IoT Cyber Attack), a new synthetically generated NIDS dataset comprising 2,298,904 records, 84 features, and 22 attack classes, offering broader attack coverage and a richer SDN-IoT traffic profile than existing benchmarks such as InSDN, BoT-IoT, ToN-IoT, and SDN-IoT.
- We formalise and implement three collaborative compression schemes Sparsity-Preserving Quantization-Aware Training (PQAT), Cluster-Preserving Quantization-Aware Training (CQAT), and Sparsity-and-Cluster-Preserving Quantization-Aware Training (PCQAT)—that resolve the mutual-cancellation problem of naïve chaining by retaining the sparsity and centroid structure produced by earlier compression stages.
- We conduct a systematic, head-to-head evaluation of these schemes across four canonical deep-learning architectures (LSTM, RNN, DNN, and CNN) and three representative attack categories (DDoS, Probe, and BFA), jointly reporting model size, inference latency, and prediction confidence so that the deployment trade-offs are made explicit.
- We show that individual clustering can reduce model size by up to 94%, but that PCQAT acts as a stabilizing scheme that recovers the predictive confidence lost under aggressive compression; in particular, CNN-PCQAT and DNN-PCQAT sustain high detection fidelity on DDoS, Probe, and BFA traffic while achieving inference latency below 0.01ms.
- We translate the experimental findings into a set of practical, deployment-oriented guidelines that map memory and latency budgets to the most suitable collaborative optimization strategy for real-time IDS on resource-constrained SDN-IoT edge devices.

The rest of the paper is organized as follows; section2 introduces the state of the art related to this topic. In section three a description of the proposed methods. Section four describes the proposed IDS based deep learning and lightweight techniques used in this work while section five provides the main obtained results. Finally, a conclusion of the proposed work is provided in section six.

II. RELATED WORK

The past few years have seen the spread of Internet of Things gadgets and the implementations of Software-Defined Networking that has made it necessary to develop effective Network Intrusion Detection Systems that are capable of working with limited resources. Deep learning models are known to have high detection rates but commonly have large computational and memory demands, which pose a challenge to use on edge devices. To address these issues, scholars have investigated a number of model compression methods, such as pruning, quantization and clustering, to develop lightweight architectures to be used in SDN-IoT settings. Lei et al. (2020) [8] deal with this by introducing P-DNN, which is an intrusion detection technique that makes use of pruning. They do this by eliminating the connections between less important weights and retraining the network, which has enabled them to lower the complexity of the model without any significant impact on its detection performance on the KDD Cup 99 dataset. A survey of these acceleration methods

is presented by Liang et al. (2021) [9], which classifies pruning and quantization methods and outlines the trade-offs between element-wise and filter-wise optimization of real-time deployment. According to Sadhwani et al. (2023) [10] vulnerability of IoT networks is their nature and they have pointed out that the devices with weak processing unit and storage are easy to be attacked by DDoS attack. They highlight the necessity to consider these hardware drawbacks in terms of creating detection models. Complex compression plans are typically a series of methods that are working together. As a quantized neural net, they suggest the Quantized Auto encoder (QAE) [11] that includes a three-step pipeline of pruning and clustering and integer quantization (uint8 and float16). Their model is particularly constructed to recreate anomaly traffic in edge devices with a much lower memory utilization. Bouayad et al. in 2024 [12] have come up with a model Lightweight-Fed-NIDS implemented in Federated Learning (FL) framework and introduced the structured model pruning approach. This allows computation of pruning mask in a single shot to optimize local models on clients, without accessing private data on the clients.

Other than pruning, quantization-based numerical optimization and architectural search are crucial. Acharya et al. (2024) [13] introduce a method of Quantization-aware Neural Architectural Search (NAS) to evolve automatically models, which are 1,000 times smaller than state-of-the-art networks. Their results show that quantized models can be very efficient in the deployment to FPGA and are also highly accurate. Rangasikunpum et al. (2025) [14] go even further and suggest Pruned Binarised Neural Networks (BNNs) to in-vehicle networks. It is an algorithm that performs banalization with 1 bit along with pruning to generate incredibly lightweight models with the ability to inquire fast on embedded platforms. Makenali et al. (2025) [15] take the synergy of the two methods and propose using similarity-based filter pruning in combination with Adaptive Power-of-two (APoT) quantization. They show that a combination or consecutive implementation of these techniques presents more compression advantages as compared to their respective independent implementation. The compression methods used in each of the references are summarized in Table I.

TABLE I.
COMPARATIVE OF RELATED WORK.

Reference	Pruning	Quantization	Clustering	Collaborative Chain
Lei et al. (2020)	✓			
Liang et al. (2021)	✓	✓	✓	✓
Sadhwani et al. (2023)				
Sharmila & Nagapadma (2023)	✓	✓	✓	✓
Bouayad et al. (2024)	✓			
Acharya et al. (2024)		✓		
Rangasikunpum et al. (2025)	✓	✓		✓
Makenali et al. (2025)	✓	✓		✓

III. METHODOLOGY

This conceptual approach of the realization of a lightweight Intrusion Detection System (IDS) specifically designed to match the Internet of Things (IoT) aims at addressing this tight security ask and limited resource availability of edge applications. Specifically, it does not mandate the computational and memory overhead of a cloud-centric architecture, but rather all software must be "frugal," meaning that they must be able to run on micro-controllers or relatively small gateways. The approach presented in the next steps will be a step-by-step guideline on how to convert a complex Deep-Learning (DL) model to a resource-aware, usable Security solution, able to meet the requirements of performance and real-life IoT environments.

IV. INTRUSION DETECTION SYSTEMS

Intrusion detection systems represent a group of software or hardware-based mechanisms designed to identify anomalies in a computer or network environment by summarizing the results of one or more sources, which are then analyzed and used to identify possible irregularities in security, and apply results to a system administrator who is authorized to create corrective measures. Academic literature has categorized IDS into anomaly and signature-based types, routinely [16]. Our next division here is Network-based IDS and Host-based IDS (HIDS): the former examines traffic at strategically vital points in the network infrastructure, whereas the latter examines system-level data limited to single machines. An integrative IDS resulting in a synergetic combination of both modalities provides an inclusive defensive perimeter and still is an unexplored field. These typologies of IDS have been systematically analyzed in Table II.

TABLE II.
COMPARATIVE ANALYSIS OF IDS PRIMARY TYPES
[16].

Type	Detection Scope	Input Data	Key Advantages	Major Limitations
HIDS	Host	Host processes, logs, config.	Analyzes encrypted data, indicates if an attack successful, no extra hardware needed.	Fails if OS is compromised, can't identify network attack, resource-intensive.
NIDS	Network	Network packets, user profiles, preceding events.	Host-independent operation, detects network-layer attacks.	Can't verify if attack success, unable to analyze encrypted traffic, limited host visibility.
Hybrid	Host + Network	According to the systems.	Flexible and efficient.	Resource-intensive, extensive duration to analyze data.

V. LIGHTWEIGHT TECHNIQUES

The frequent lack of edge device computing power to handle complex pipelines inference tasks is a pressing requirement to develop and deploy lightweight, efficient

models that will ease those constraints. Model pruning based optimization methods, quantization-based optimization methods, weight clustering-based optimization methods and collaborative filtering-based optimization methods have become the most popular in the current literature as they enable the reduction of the memory footprint and computation load of deep transfer learning models. These methods have collateral advantages, such as high accessibility, improved preservation of privacy, and decreased cost of operation, but might involve a measurable decrease in the overall inference pace. The major drawbacks of lightweight schemes are advanced algorithms, lower predictive accuracy, poor flexibility, and unavoidable trade-off between model size, speed and accuracy.

The pruning, as an optimization method, is a systematic procedure used on a neural network to remove redundant neurons, unconnected connections, insignificant weights, and/or unnecessary parameters in a neural network to decrease the number of parameters [17]. To reduce the memory usage of the model, the weights whose magnitude is below a certain threshold can be removed from the model. Empirically, after applying the pruning step, a re-trained model can be better than the one trained before; this fact may be explained by the fact that useless/noisy connections are removed. The other important part of model optimization is quantization. The storage and arithmetic needs of a deep learning model can be lowered by storing high precision weight values on low precision representations, such as 16-bit or 32-bit fixed point arithmetic [18]. Binary floating-point formats can also be used to even more cost-effectively minimize computational resources. There are two common methods of quantization: post training quantization (PTQ), which assigns less precise values without retraining the model; and quantization aware training (QAT), which fine tunes the model so that it can be trained with lower-precision values. In the area of statistics, "weight clustering" or "weight sharing" is a way of determining representative values of sets of weights, thereby minimizing the number of distinct parameters that are needed to be stored or computed [19]. This method is especially useful in decreasing the parametric diversity within a network; it can be said that it is possible to use just one single weight representation to represent numerous neurons, without decreasing representational capability.

Collaborative optimization is a universal system that integrates, applications of multiple optimizations to ensure a balance between inference speed, model size and predictive power are optimized [20][21]. The concept of using sequentially the basic methods of pruning, quantization and clustering, so that the total benefits are received where deployed. There is an issue with the naive concatenation of these optimizations, namely that one of these operations could have side effects that nullify the effect of the other. For instance, sparsity may be introduced by pruning and this can be mitigated by clustering. The 3 hypothesis that we propose to reduce this effect are experimental variations of clustering-based optimization: (1) Sparsity -Preserving Quantization - Aware Training (PQAT), (2) Cluster -Preserving Quantization (CQAT) and (3) Sparsity -and-Cluster -Preserving Quantization -Aware Training (PCQAT) [22]. Both approaches are meant to conserve the structural advantages of the previous steps as well as compressing the

model further. Fig. 1 shows several deployment pathways that enable practitioners to take the path from a raw model to the optimized artefact, ready for deployment. The graph leaf nodes have models that are quantized partially or fully and are exported in TFLite. The labels at the nodes are the optimization technique used to get to the state. Cumulative optimization the direct, quantization-only path - either PTQ or QAT - is not shown in the diagram in order to underscore the benefits of cumulative optimization.

The general objective is to achieve a deployed terminal level of the deployment tree that is fully optimized. Yet, with the latency and accuracy constraints of the application, the lower-level needs might be already met by the intermediate level in line with the operational specifications. Practically, one works in the direction of the right branches of the optimization tree, at each step paying attention to the fact that the model achieves the required latency. In the event that the model does not satisfy these requirements the collaborative optimization process is used to obtain further compression and the process is repeated until the model has settled on the desired equilibrium between pruning, clustering, and quantization. Fig. 2 illustrates density plots of typical weight kernels that are computed empirically in the course of the optimization pipeline and which demonstrate the growing sparsity and the reduced cardinality of values as the outcome of collaborative optimization. These visualizations ensure that the deployed model has a significantly slimmer weight distribution and a smaller number of distinct parameter values which allows specialized hardware to take advantage of sparsity and grouped structures to further reduce latency. Algorithms 1,2, and 3 shows the PQAT, CQAT, and PCQAT techniques respectively.

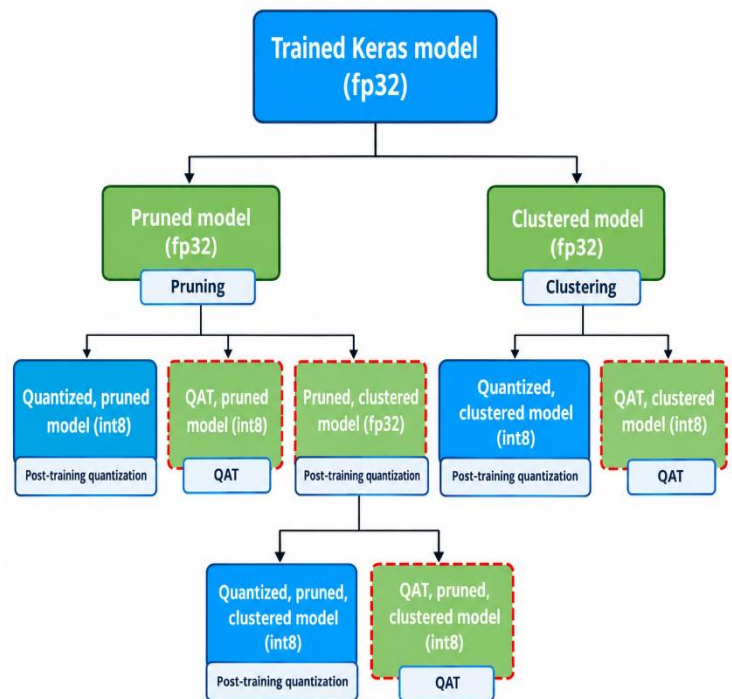


Fig.1. Deployment pathways of collaborative lightweight techniques.

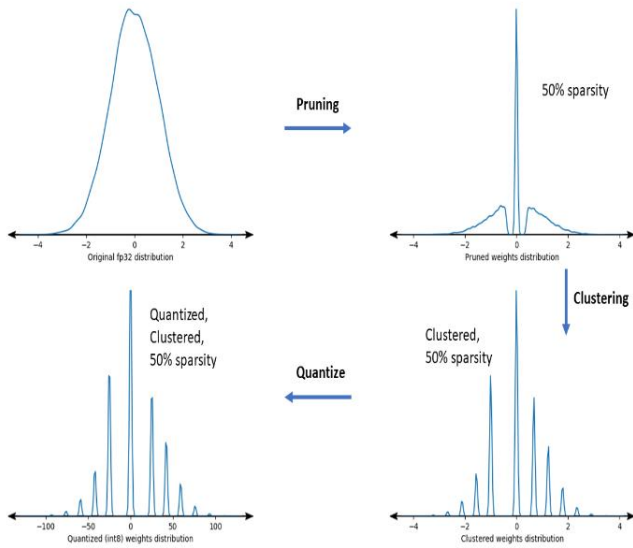


Fig.2. A visualization of the sequential stages of model compression: Pruning to remove redundant weights (creating 50% sparsity), Clustering to group similar weight values together, and Quantization to convert high-precision (fp32) values into efficient 8-bit integers (int8).

Algorithm 1: Pruning-preserving QAT (PQAT)

Input: model weight

Output: Pruning preserving quantization model

Step1: load pretrained model

Step2: set pruning hyperparameters

Prun percentage, initial_sparsity, final_sparsity,

step3: tune the pruned model

step4: strip the pruned model to remove temporary weights

step5: set quantization aware hyperparameters

num_bits = 16

quantize_range = $[-2^{(\text{num_bits}-1)}, 2^{(\text{num_bits}-1)} - 1]$

step6: Apply Quantization to Weights and Activations:

scale_factor_weights = $\max(\text{abs}(\text{layer_weights})) / (2^{(\text{num_bits}-1)})$

scale_factor_activation = $\max(\text{abs}(\text{layer_activation})) / (2^{(\text{num_bits}-1)})$

step7: Final Quantization of Weights and Activations:

for each layer in model:

scale_factor_weights = $\max(\text{abs}(\text{layer_weights})) / (2^{(\text{num_bits}-1)})$

layer_weights = $\text{quantize}(\text{layer_weights}, \text{scale_factor_weights})$

scale_factor_activation = $\max(\text{abs}(\text{layer_activation})) / (2^{(\text{num_bits}-1)})$

layer_activation = $\text{quantize}(\text{layer_activation}, \text{scale_factor_activation})$

step8: tune the pruned- quantized model

step9: strip the pruned- quantized model to remove temporary weights

step10: apply PQAT and observe model clusters and sparsity

step11: retrain the PQAT model

step12: End

Algorithm 2: Cluster preserving quantization aware training (CQAT)

Input: model weight

Output: Cluster preserving quantization model

Step1: Load Pretrained Model

Step2: set cluster hyperparameters:

Number of weight clusters, Number of clusters

step3: tune the clustered model

step4: strip the clustered model to remove temporary weights

step5: set quantization aware hyperparameters

num_bits = 16

quantize_range = $[-2^{(\text{num_bits}-1)}, 2^{(\text{num_bits}-1)} - 1]$

step6: Apply Quantization to Weights and Activations:

scale_factor_weights = $\max(\text{abs}(\text{layer_weights})) / (2^{(\text{num_bits}-1)})$

scale_factor_activation = $\max(\text{abs}(\text{layer_activation})) / (2^{(\text{num_bits}-1)})$

step7: Final Quantization of Weights and Activations:

for each layer in model:

scale_factor_weights = $\max(\text{abs}(\text{layer_weights})) / (2^{(\text{num_bits}-1)})$

layer_weights = $\text{quantize}(\text{layer_weights}, \text{scale_factor_weights})$

scale_factor_activation = $\max(\text{abs}(\text{layer_activation})) / (2^{(\text{num_bits}-1)})$

layer_activation = $\text{quantize}(\text{layer_activation}, \text{scale_factor_activation})$

step7: retrain the CQAT model

step8: Save the CQAT Model

step9: End

Algorithm 3: sparsity and cluster preserving QAT (PCQAT)

input: model weight

Output: Sparsity and cluster preserving model

Step1: Load Pretrained Model

Step2: set pruning hyperparameters

Prun percentage, initial_sparsity, final_sparsity,

step3: Fine-tune the pruned model

step4: strip the pruned model to remove temporary weights

step5: Apply sparsity preserving clustering :

1-set clustering hyperparameters Number of weight clusters

2- compute weight centroid:

cluster_centers = initialize_random_centers (weights, num_clusters)

for each weight in weights:

closest_center = find_nearest_center (weight, cluster_centers)

weight_clusters [closest_center].append(weight)

step6: Fine-tune the clustered- pruned model

step7: strip the clustered- pruned model to remove temporary weights

step8: Apply QAT and check sparsity

step9: Fine-tune the QAT model

step10: Apply PCQAT and check sparsity

step11: Fine-tune the PCQAT model

step12: End

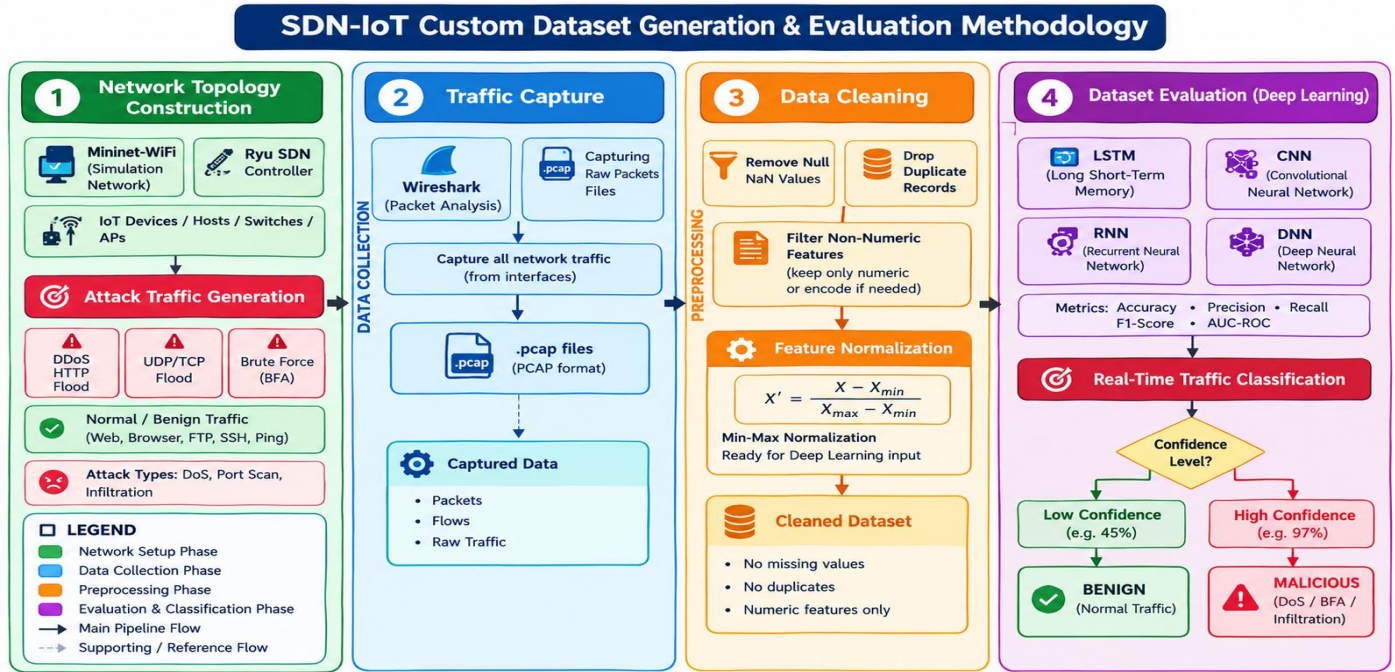


Fig. 3. Proposed costume SICA dataset generation and preprocessing.

VI. SICA DATASET

In this work, a costume dataset has been generated to test the proposed NIDS. Fig.3 presents a comprehensive end-to-end pipeline for generating a custom network intrusion detection dataset within a simulated SDN-IoT environment, and subsequently using that dataset to train and evaluate multiple deep learning models. The entire methodology is organized into four sequential phases, each building upon the output of the previous one, forming a coherent and reproducible research workflow:

- Network setup phase:** environment, the basis for the experiment. We build a virtual network based on Mininet-WiFi, controlled by the RYU SDN controller, and include a number of simulated IoT devices along with hosts, switches and access points. In this network, a certain amount of traffic is injected. The first is attack traffic based on some of the most common attack types including Distributed Denial of Service (HOIC HTTP Flood and LOIC UDP/TCP Flood), Brute Force Attack (Hping3 BFA/Syn), port scanning, Denial of Service (DoS) and infiltration attacks. The second is benign traffic that corresponds to legitimate traffic such as web browsing, FTP, SSH, and Ping. This ensures that the data set is comprised of a representative mix of malicious and benign traffic.
- Data collection phase:** where the network packets flowing in the simulated environment are collected and converted to a structured digital format. The packet analyzer of choice is Wireshark to record the raw network traffic in the .pcap and .pcapng file formats, capturing all network interfaces. These raw packet files are then fed as input to CICFlowMeter, a meter to extract flow-level features that generates about 80 features from

each network flow based on various statistical measures such as flow duration, bytes per packet, inter-arrival time, and protocol flags. Each flow is also labeled either as benign or a particular type of attack (such as DDoS and BFA). At the end of the process, a structured set of features (featureset.csv file) is produced in which each row corresponds to a network flow and each column to a feature.

- Dataset preprocessing:** to make the original dataset suitable for the input of deep learning in two steps. The first one is data cleaning, where the data set is cleaned from null/NaN values, duplicate records, and non-numeric features, leaving only the numeric columns. Next is feature normalization, which involves scaling all feature values to a range of 0-1 using Min-Max normalization $X' = (X - X_{min}) / (X_{max} - X_{min})$. This step rules out a situation where individual features' contributions to the learning process vary due to different scales and yields a clean, normalized feature set that is ready for input into deep learning models.
- Evaluation and classification phase:** In which the pre-processed data are fed into four different deep learning models: LSTM, CNN, RNN, and DNN. The models' evaluation is done with the metrics of Accuracy, Precision, Recall, and F1-Score, enabling a comprehensive comparative performance analysis of the models' capabilities to detect the network trafficking intrusion. The figure also showcases a decision-making process for real-time traffic classification based on confidence levels. A low-confidence result from the model will result in benign or "normal" classification of the traffic (i.e. non-representative feature extraction). On the other hand, if the model is confident in its prediction, the traffic will be classified as malicious, such as DDoS, BFA and Infiltration attacks. Table III shows the models hyper parameters.

TABLE III
DIFFERENT MODELS HYPER PARAMETERS.

Model	Architecture	Neurons per layer	Hidden activation	Output activation	Other parameters
LSTM	Input $\rightarrow$ LSTM $\times$ 5 $\rightarrow$ Output	256 $\rightarrow$ 128 $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 32 $\rightarrow$ Output	Tanh	Softmax	—
CNN	Input $\rightarrow$ (Conv1D + MaxPool) $\times$ 3 $\rightarrow$ Flatten $\rightarrow$ Dense $\times$ 2 $\rightarrow$ Output	256 $\rightarrow$ 128 $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 32 $\rightarrow$ Output	ReLU	Softmax	Kernel size: 2, 3
RNN	Input $\rightarrow$ GRU $\rightarrow$ Dense $\times$ 2 $\rightarrow$ Output	64 $\rightarrow$ 64 $\rightarrow$ 32 $\rightarrow$ Output	ReLU	Softmax	—
DNN	Input $\rightarrow$ Dense $\times$ 5 $\rightarrow$ Output	256 $\rightarrow$ 128 $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 32 $\rightarrow$ Output	ReLU	Softmax	—

Overall, this framework represents a completely simulation-based approach to producing a labeled network intrusion detection dataset, without the need for any hardware. Through the integration of an SDN-IoT based realistic network simulation, efficient data collection, thoughtful data preprocessing and robust deep learning evaluation, the framework offers a fully reproducible methodology for assessing and benchmarking intrusion detection systems for modern IoT networks. Table IV shows statistical analysis of the SICA (SDN-IoT Cyber Attack) dataset implementation compared against other datasets SDN-IoT [2] and InSDN [5].

TABLE IV.
SICA DATASET COMPARISON AGAINST INSDN AND SDN-IOT

Parameter	InSDN	SDN-IoT	SICA
Record	343,989	844,242	2,298,904
Feature	83	84	84
Attack classes	9	14	22
Size	141.71 MB	470.31 MB	1,262 MB
Network type	SDN	SDN + IoT	IoT

VII. PROPOSED COLLABORATIVE TECHNIQUES

Fig.4 briefly outlines an extensive architecture of building and refining a deep-learning-based intrusion detection system (IDS) that would best fit software-defined networking (SDN) and Internet of Things (IoT) environments. The process is divided into three cardinal steps that include data preprocessing, model training and evaluation, and collaborative model optimization. It starts with SICA dataset, which includes 84 attributes of network-traffic that are relevant to SDN-IoT scenarios. Before this data can be used to train, it has to pass through three essential preprocessing stages namely cleaning, normalization and labelling. Cleaning removes noise, missing entries and inconsistencies in the raw traffic logs. Normalization shifts the values of features to a similar scale, which is a requirement of deep-learning convergence to a stable and efficient point. Identifying a class identifier is allocated to every instance through the labeling to identify normal traffic and different

manifestations of intrusion and attacks. All these steps will ensure that data fed into the model is of adequate quality to undergo supervised learning.

After preprocessing, the corpus is split by a 70/30 split (70 percent gates are used to train the model and a slice of the same set is used as validation data when training a model), and the remaining 30 percent is used as test data. Such an allocation plan helps reduce over-fitting and provides an unbiased evaluation of the final artefact. This training on this curated data continues with the deep-learning architecture, which is visualized in the upper right as a multilayer neural network. Once convergence is reached, formal evaluation routine measures the baseline performance in terms of traditional metrics, including, but not limited to, accuracy, precision, recall, and F1 -score.

The most technically subtle part of the diagram lies in the middle zone where four parallel optimization pipelines, each representing a unique collaboration compression paradigm, are shown. They all have a common ancestor: a 32-bit floating-point-trained baseline model. This point onwards, every branch implements a unique blend of compression measures. The former process starts with model pruning, continues with sparsity preserving clustering on the pruned model, and concludes with a combination of Pruning-Aware Clustering Quantization-Aware Training (PCQAT) with quantization-aware training on the pruned artefact. The second pathway starts with clustering (8 clusters), followed by stripping of the clustered model, and runs both Cluster Preserving Quantization Aware Training (CQAT) and quantization aware training on the stripped one. The third scheme gives the priority to pruning which is then followed by the quantization-aware training of the pruned model followed by the pruning-preserving quantization-aware training (PQAT). The fourth path combines all the three maneuvers sequentially, that is, pruning, stripping, clustering (8 clusters), and eventual sparsity-preserving clustering, thus, representing the most comprehensive collaborative compression methodology. As the four streams of optimization finish their compression and quantization steps, the four models all meet together at a single retraining step, which is indicated by a salmon-colored bar. This retraining refines the compressed networks to correct any accuracy loss incurred at the process of optimization. Lastly, the retrained models will go through a shared graduated module that will be found at the bottom of the diagram, where the performance and computational efficiency of the models are compared.

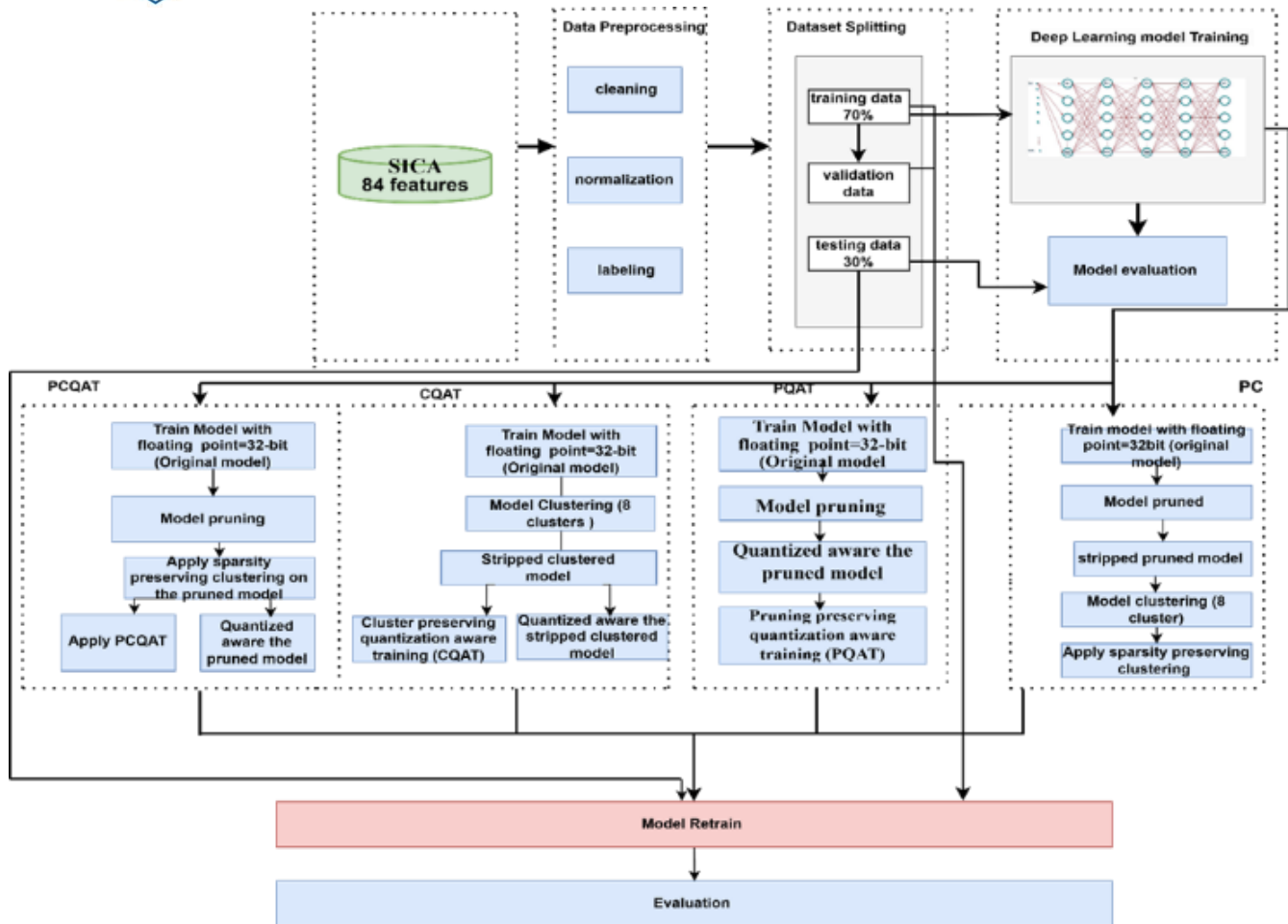


Fig. 4. Proposed deep learning pipeline with model optimization techniques.

Such comparative evaluation provides the researchers with the information they require to ascertain which joint approach provides the best trade-off between model faithfulness and resource-economy, an imperative of applying IDS solutions on bandwidth-limited and compute-restricted SDN-IoT backbones.

Overall, the figure presents a procedural and comprehensive roadmap towards the installation of a lightweight but precise intrusion detection framework by questioning several compression pathways in unison, thus making the solution supremely feasible in the real-world implementation in bandwidth-sensitive, hardware-constrained SDN-IoT systems.

VIII. RESULTS AND DISCUSSION

The proposed IDS is tested on four modern deep-learning architectures, LSTM, RNN, DNN and CNN, with three light weight compression methods: pruning, clustering and quantization. There are 3 categories of attacks that are considered: DDoS, Probe, and BFA. The proposed system is tested using a traffic sample of 300 packets, with consistent results for all the repetitions, which are averaged to get the values reported in Table V and VI. The results of each

compression technique alone are presented in Table VI. The most significant reduction in model size across all networks is achieved when clustering: The LSTM model is reduced from 35.27 KB to 2.12 KB, which is approximately a 94% reduction. Likewise, under clustering, the heaviest configuration (DNN) reduces from 55.71 KB to 17.4 KB and

under quantization it reduces to 19.9 KB. Pruning is less aggressive in decreasing the footprint but retains a higher percentage of the original weight structure. The RNN is the smallest of the baseline (13.98 KB), and compressing it further supports its minimal footprint. Confidence scores reflect the level of certainty that the model has about their prediction of the class: A higher score corresponds to a higher level of confidence. Confidences for both LSTM and RNN networks remain strong even with compression—>1.06 across all compression techniques and for all attacks, and occasionally even improve with quantization. In comparison, the DNN has the highest degradation, training with a baseline confidence of 1.62, but this confidence drops significantly after pruning, clustering or quantization to approximately 1.06. The CNN exhibits a quite different behavior: The clustering-induced confidence of 1.77 is higher than the uncompressed confidence of 1.05, likely due to the fact that clustering serves as a kind of regularization to remove redundant kernels and preserve the most discriminative ones. Fig. 5 compares the accuracy, loss, and model size of the DNN, LSTM, RNN, and 1D-CNN models when pruning, clustering, and quantization are applied individually. Clustering gives the most consistent outcome: all four models retain an accuracy of about 0.96–0.97 while reaching the smallest footprints, from 7.65 KB for the 1D-CNN to 17.67 KB for the LSTM. Pruning preserves the accuracy of the

DNN (0.961) but degrades the recurrent and convolutional models, most notably the 1D-CNN, whose accuracy falls to 0.737, and it produces larger models of 10.81–27.05 KB. Quantization of the DNN yields an 18.

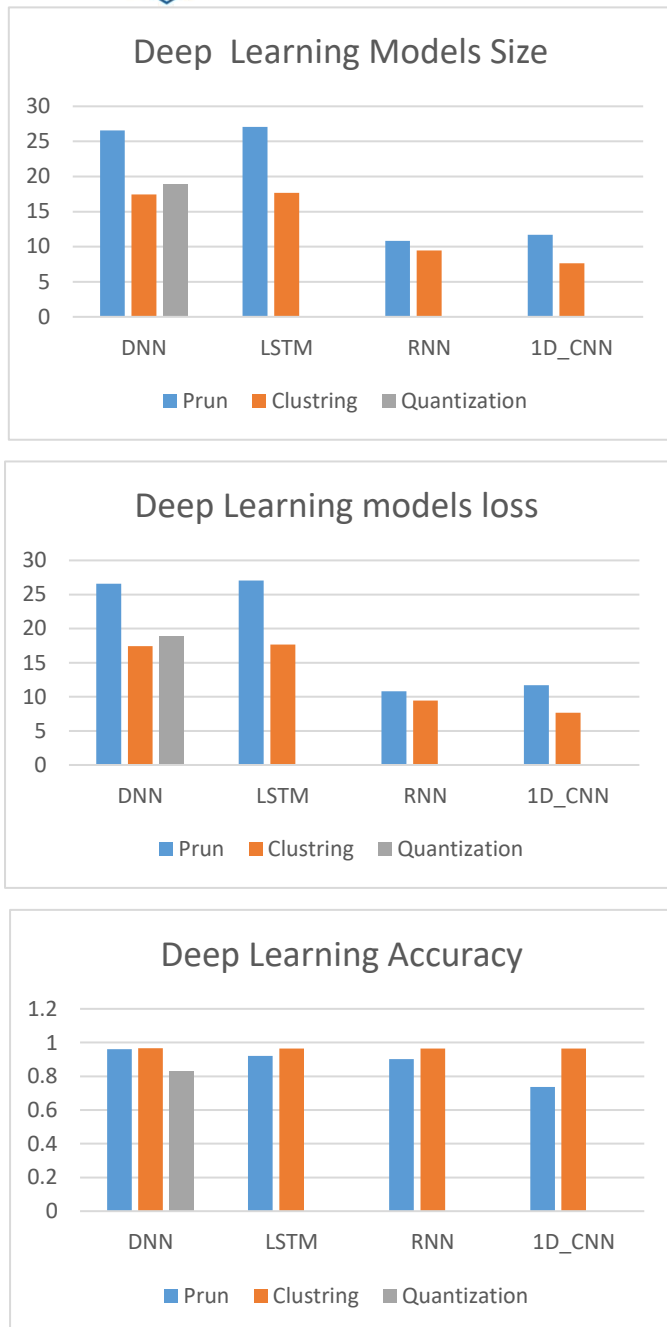


Fig 5. Different deep learning lightweight accuracy, loss, and model size

KB model with an accuracy of 0.832, indicating that reducing numerical precision alone costs more accuracy than weight sharing at a comparable size. These trends confirm that clustering offers the best single-technique balance between compactness and detection performance.

The results of the inference time show an unexpected trade-off. The fastest inference is achieved in case of quantization where the computational cost of the LSTM is reduced to 2.50ms due to the lower numerical precision and computational cost. In contrast, the pruning process itself can actually result in a significant increase in inference time, such as the pruned LSTM, which takes 27.46ms, because most standard deep-learning runtimes do not leverage sparse weight matrices: even if there are zero weights, they still have to fit into dense memory layouts, which means they don't necessarily speed up the run. The inference times for clustering are moderately dependent on model architecture.

The DDoS, Probe and BFA attack categories do not add significantly to detection performance, however surrogate traffic (such as spoofed traffic or traffic impersonation) sometimes leads to lower confidence scores, especially when using the baseline CNN and DNN models. This fact suggests that the recognition of an adversarial or camouflaged attack is more difficult in general and that lightweight models must be carefully tested against evasion-type attacks. A thorough performance analysis of the three collaborative quantization-aware training methods PQAT, CQAT and PCQAT is presented for the four deep-learning architectures employed for intrusion detection in Table V.

Based on these data, we study the interaction of model compactness, inference latency, and classification accuracy that occurs under this light weight optimization paradigm.

One of the most remarkable findings is the high level of disparity between the efficiencies of the model families. Compared to the CNN and RNN-based networks, which have the smallest model sizes of up to 14.53 KB and inference time of 0.004ms to 0.010ms, the LSTM-based networks have significantly larger model sizes and forward-propagation times of up to 41 KB and up to 0.23ms, respectively.

The CNN and RNN configurations generated with these QAT strategies result in a clear advantage in high throughput applications in the context of real-time IDS deployments, where even a few additional milliseconds can cause dropped packets. The three collaborative schemes produce a smaller model footprint than PQAT in both instances, with CQAT and PCQAT consistently yielding smaller footprints than PQAT. For the CNN model, for instance, PQAT produces a 16.1 KB model while the CQAT and PCQAT models are both compressed to about 14.5 KB, demonstrating the power of cluster-based compression. This extra compression, however, tends to be coupled by low prediction confidence.

The confidence levels of LSTM become unchanging across the three schemes, whereas for RNN and CNN, the visibly decreasing confidence levels drop into a range of less than 0.40 when PQAT is replaced by the more aggressive CQAT. The combination of the "footprint" that CQAT provides with the "structure preserving" mechanism of PCQAT produces a good stabilizer.

One example is the CNN row: PQAT classifies the sample as BFA with a confidence of 0.80, while PCQAT does it with higher confidence of 0.88 and with a smaller model. In this evidence, the fact that both sparsity and cluster structure are maintained together reduces the loss of discriminative features that occurs in single-stage compression schemes. Last but not least, the selection of the QAT scheme should depend on the target deployment environment.

The CNN-CQAT variant has minimum memory requirements and the fastest response time for edge devices that have limited memories. On the other hand, DNN-PCQAT and CNN-PCQAT ensure the best trade-off between attack labelling reliability and inference latency when the criterion is the ability to detect the attacks accurately while adhering to a tight overhead budget and satisfying the confidence level needed for the subsequent defensive measures.

TABLE V.
INDIVIDUAL LIGHTWEIGHT TECHNIQUES RESULTS FOR 300 PACKET

Attack Category	LSTM (base)				LSTM Prune				LSTM Cluster				LSTM Qat		
	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence
BFA	68.27	BFA	0.8219	0.06 ms	27.046	DDoS	0.798335	0.0925	17.669	DDoS	0.719197	0.097	-	-	-
DDoS	68.27	DDoS	0.9656	0.06 ms	27.046	DDoS	0.78747	0.09325	17.669	DDoS	0.86546	0.10365	-	-	-
Probe	68.27	Probe	0.2956	0.26 ms	27.046	Probe	0.41147	0.087018	17.669	DDoS	0.491566	0.093158	-	-	-
Attack Category	RNN (base)				RNN Prune				RNN Cluster				RNN Qat		
	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence
BFA	18.96	BFA	0.7303	0.06 ms	10.812	DDoS	0.626135	0.0705	9.459	DDoS	0.905968	0.1505	-	-	-
DDoS	18.96	DDoS	0.9669	0.06 ms	10.812	DDoS	0.707992	0.07605	9.459	DDoS	0.905091	0.11185	-	-	-
Probe	18.96	Probe	0.3414	0.25 ms	10.812	Probe	0.546567	0.072281	9.459	DDoS	0.623057	0.107193	-	-	-
Attack Category	DNN (base)				DNN Prune				DNN Cluster				DNN Qat		
	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence
BFA	80.71	BFA	0.8212	0.06 ms	26.586	DDoS	0.850372	0.005	17.423	BFA	0.849724	0.0055	18.808	[http_flood]	0.715332
DDoS	80.71	DDoS	0.9802	0.06 ms	26.586	DDoS	0.807679	0.0071	17.423	DDoS	0.821406	0.00475	18.808	[http_flood]	0.796465
Probe	80.71	udp_reflection	0.7002	0.31 ms	26.586	DDoS	0.528366	0.005439	17.423	Probe	0.703944	0.004912	18.808	[apache_reflection]	0.37685
Attack Category	CNN (base)				CNN Prune				CNN Cluster				CNN Qat		
	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence
BFA	22.43	BFA	0.8432	0.06 ms	11.701	DDoS	0.380788	0.0115	7.648	BFA	0.750968	0.0105	-	-	-
DDoS	22.43	DDoS	0.9545	0.07 ms	11.701	DDoS	0.379386	0.0148	7.648	DDoS	0.774739	0.0104	-	-	-
Probe	22.43	BFA	0.7728	0.24 ms	11.701	DDoS	0.351902	0.010877	0.010877	DDoS	0.416424	0.01	-	-	-

TABLE VI.
LIGHTWEIGHT TECHNIQUES CHAINING RESULTS FOR 300 PACKET

LSTM PQAT				LSTM CQAT				LSTM PCQAT				
Attack Category	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)
BFA	41.66	apache_range_dos (DDoS)	0.953807	0.14975	44.85	http_get_uri_strings (DDoS)	0.91875	0.22	44.85	http_get_uri_long (DDoS)	0.943555	0.20875
DDoS	41.66	ntp_prot (DDoS)	0.802061	0.162727	44.85	ntp_prot (DDoS)	0.766472	0.232238	44.85	ntp_prot (DDoS)	0.838287	0.207273
Probe	41.66	udp_reflection (DDoS)	0.712511	0.157544	44.85	teardrop (DDoS)	0.638295	0.235263	44.85	teardrop (DDoS)	0.387678	0.212456
RNN PQAT				RNN CQAT				RNN PCQAT				
Attack Category	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)
BFA	16.14	http_get_uri_strings (DDoS)	0.848962	0.0105	14.62	http_get_uri_long (DDoS)	0.735254	0.01125	14.62	http_get_uri_long (DDoS)	0.77998	0.01
DDoS	16.14	http_get_uri_strings (DDoS)	0.731956	0.01014	14.62	apache_range_dos (DDoS)	0.721564	0.01021	14.62	ntp_prot (DDoS)	0.676737	0.010629
Probe	16.14	http_get_uri_long (DDoS)	0.670966	0.010526	14.62	http_get_uri_long (DDoS)	0.394052	0.010351	14.62	ddos	0.511445	0.010702
DNN PQAT				DNN CQAT				DNN PCQAT				
Attack Category	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)
BFA	25.08 KB	http_flood / http_get_uri_long (DDoS)	0.649609	0.01075	30.15 KB	http_get_uri_strings (DDoS)	0.953027	0.0105	30.15 KB	http_get_uri_long (DDoS)	0.842285	0.01075
DDoS	25.08 KB	http_flood (DDoS)	0.705283	0.01007	30.15 KB	http_flood (DDoS)	0.72241	0.01007	30.15 KB	http_get_uri_strings (DDoS)	0.766417	0.011469
Probe	25.08 KB	http_flood (DDoS)	0.55222	0.010702	30.15 KB	apache_range_dos (DDoS)	0.657141	0.010702	30.15 KB	http_get_uri_strings (DDoS)	0.565515	0.011579
CNN PQAT				CNN CQAT				CNN PCQAT				
Attack Category	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)	Size (KB)	Classified Label	Confidence	Time (ms)
BFA	16.1	normal	0.809551	0.00475	14.53	http_get_uri_long (DDoS)	0.765137	0.01	14.61	http_get_uri_long (DDoS)	0.888184	0.01075
DDoS	16.1	http_get_uri_strings (DDoS)	0.757584	0.005245	14.53	http_get_uri_strings (DDoS)	0.738937	0.01014	14.61	http_get_uri_long (DDoS)	0.764778	0.01
Probe	16.1	http_get_uri_long (DDoS)	0.776185	0.007018	14.53	http_get_uri_long (DDoS)	0.410704	0.010351	14.61	http_get_uri_long (DDoS)	0.472108	0.010526

IX. CONCLUSION

The focus of this study was on the problem of deploying powerful IDS in the resource-constrained environments of SDN-IoT. We performed a study on the interaction of pruning, clustering, and quantization and demonstrated that collaborative optimization of the three can remove the drawbacks of "naive" sequential chaining, in which one compression method will often counteract the gains of another. From our experimental study, we have gained the following insights:

- Resilience to compression: We found that RNN and LSTM models performed particularly well in terms of confidence score stability across optimization paths.
- In our experiments, we found that PCQAT, which we propose as a new Stabilizer scheme, was the most balanced compression strategy to mitigate the loss of discriminative features that is found in single-stage schemes.
- Operational Efficiency: The introduction of quantization-aware techniques greatly lowered inference time, with CNN and RNN variants reaching response times as low as 0.004ms, making them ideal for high-throughput real-time applications.

For deployment, the optimization method should depend on the particular constraint in the deployment environment. CQAT variants provide the smallest memory footprint for devices that have low memory resources, whereas PCQAT gives a better footprint/confidence ratio when high-confidence detection is needed. The proposed framework takes a step towards the deployment of an efficient deep-learning based IDS on resource constrained SDN-IoT devices within the architectures, attack type and dataset scope examined in this study and further validation for larger scale deployments and other attack categories is left as future work.

References

- [1] Z. Wang, H. Chen, S. Yang, X. Luo, D. Li, and J. Wang, "A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization", *PeerJ Computer Science*, 9, 2003. <https://doi.org/10.7717/peerj-cs.1569>
- [2] H. Dhirar and A. Hamad, "Comparative evaluation of a novel IDS dataset for SDN-IoT using deep learning models against InSDN, BoT-IoT, and ToN-IoT", *Measurement Digitalization*, vol. 4, 100015, 2025. <https://doi.org/10.1016/j.measdig.2025.100015>
- [3] "The bot-IoT dataset | UNSW Research." Accessed: May 27, 2025. [Online]. Available: <https://research.unsw.edu.au/projects/bot-iot-dataset>.
- [4] "ToN_IoT datasets | IEEE DataPort." Accessed: Nov. 24, 2024. [Online]. Available: <https://iee-dataport.org/documents/toniot-datasets>
- [5] M. S. Elsayed, N. A. Le-Khac, A. D. Jurcut, "InSDN: a novel SDN intrusion dataset", *IEEE Access*, vol. 8, 2020. <https://doi.org/10.1109/ACCESS.2020.3022633>
- [6] H. Dhirar, and A. Hamad, "Internet of Things Software-Defined Network Intrusion Detection Dataset: Inter- and Intra-Domain", *Al-Khwarizmi Engineering Journal*, vol. 21, no.3, pp. 35–47, 2025. <https://doi.org/10.22153/kej.2025.08.001>
- [7] M. Fatima, O. Rehman and I. M. H. Rehman, "Li-IDS: An Approach Towards a Lightweight IDS for Resource-Constrained IoT," 2023 International Conference on Smart Applications, Communications and Networking (SmartNets), Istanbul, Turkey, 2023, pp. 1-6, doi: 10.1109/SmartNets58706.2023.10216096
- [8] M. Lei, X. Li, B. Cai, Y. Li, L. Liu and W. Kong, "P-DNN: An Effective Intrusion Detection Method based on Pruning Deep Neural Network," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-9, doi: 10.1109/IJCNN48605.2020.9206805
- [9] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey", *In Preprint Submitted to Elsevier*, vol.461, pp. 370-403, 2021. doi:10.1016/j.neucom.2021.07.045
- [10] S. Sadhwani, B. Manibalan, R. Muthalagu, and P. Pawar, "A lightweight model for DDOS attack detection using machine learning techniques". *Appl. Sci.*, vol. 13, issue 17, 2023. <https://doi.org/10.3390/app13179937>
- [11] B. S. Sharmila, and R. Nagapadma, "Quantized autoencoder (QAE) intrusion detection system for anomaly detection in resource-constrained IoT devices using RT-IoT2022 dataset". *Cybersecurity*, vol. 6, 2023. <https://doi.org/10.1186/s42400-023-00178-5>
- [12] A. Bouayad, H. Alami, M. J. Idrissi, I. Berrada, "College of Computing, Mohammed VI Polytechnic University (UM6P), Laboratory of Informatics, Signals, Automatics, and Cognitivism (LISAC), Faculty of Sciences Dhar El Mahraz, Sidi Mohammed Ben Abdellah University, & Abdelhak Bouayad. (2024). Lightweight federated learning for efficient network intrusion detection. In S. D. Campbell (Ed.), *ACCESS*, Vol. 12, p. 172027, <https://doi.org/10.1109/ACCESS.2024.3494057>
- [13] R. Yu, L. L. Jeune, and N. Mentens, "University of Florida, KU Leuven, & Worcester Polytechnic Institute. (2024). Quantization-aware neural architectural search for intrusion detection". *arXiv*. <https://arxiv.org/pdf/2311.04194>
- [14] A. Rangasikunpum, S. Amiri, and L. Ost, "Ultra-Lightweight and highly efficient pruned binarised neural networks for intrusion detection in In-Vehicle networks". *Electronics*, vol. 14, no. 9, p. 1710, 2025. <https://doi.org/10.3390/electronics14091710>
- [15] S. Makenali, B. Rokh, and A. Azarpeyvand, "Integrating Pruning with Quantization for Efficient Deep Neural Networks Compression", *arXiv*, 2025. <https://arxiv.org/abs/2509.04244v1>
- [16] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges", *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019, doi: 10.1186/S42400-019-0038-7/FIGURES/8
- [17] S. Han, H. Mao, w. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding". In *iclr*, 2016. <https://arxiv.org/pdf/1510.00149>
- [18] B. Jacob, S. Kligys, B. Chen, et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," 2018 IEEE/CVF Conference on Computer Vision and Pattern

- Recognition, Salt Lake City, UT, USA, 2018, pp. 2704-2713, doi: 10.1109/CVPR.2018.00286
- [19] M. Zhang, Y. Ma, J. Li, and J. Zhang, "A density connection weight-based clustering approach for dataset with density-sparse region", *Expert Systems with Applications*, vol. 230, 2023. <https://doi.org/10.1016/j.eswa.2023.120633>
- [20] H. Hussain, P. S. Tamizharasan, and P. K. Yadav, "LCRM: Layer-wise complexity reduction method for CNN model optimization on end devices", *IEEE Access*, 11, pp. 66838–66857, 2023. <https://doi.org/10.1109/access.2023.3290620>
- [21] B. Predić, U. Vukić, M. Saračević, D. Karabašević, and D. Stanujkić, "The possibility of combining and implementing deep neural network compression methods", *Axioms*, vol. 11, no. 5, 2022. <https://doi.org/10.3390/axioms11050229>
- [22] "Collaborative Optimization." TensorFlow, Google, www.tensorflow.org/model_optimization/guide/combine/collaborative_optimization. Accessed 27 Feb. 2026.