An Energy-efficient Intrusion Detection Offloading Based on DNN for Edge Computing

João A. Simioni, Eduardo K. Viegas, Altair O. Santin, and Everton de Matos

***Abstract*—To improve the accuracy of Deep Neural Networks (DNNs) applied to Network Intrusion Detection Systems (NIDS) researchers often increase the complexity of their designed model. Given the processing limitations of resource-constrained devices, researchers have proposed offloading the NIDS task to the cloud. However, simultaneously ensuring energy efficiency and detection accuracy remains a challenge. This paper proposes a DNN-based NIDS through early exits that operate following an energy-efficient edge-computing architecture implemented twofold. Firstly, we propose a DNN-based NIDS employing multi-objective optimization for efficient inference and computation offloading. It is designed to perform the classification task at the edge device and configured to proactively offload events to the cloud when additional processing capabilities are required. Our insight is to utilize multi-objective optimization to identify the optimal balance between accuracy and energy efficiency in task offloading. Secondly, the final DNN branch performs classification with a reject option to ensure reliability when analyzing new network traffic patterns, while calibration adjusts the model's confidence values to enhance generalization. The rejection mechanism allows the model to accept only its most confident classifications enhancing the model's generalization capabilities. Experiments conducted with our proposal's prototype through a new intrusion dataset encompassing one-year-long network traffic with over 7TB of data attested to our proposal's feasibility. It can reduce the edge device's energy consumption and processing costs to only 1%, while maintaining accuracy. This is achieved while demanding the offloading of only 10% of network events to the cloud, optimizing resource utilization across both the edge and cloud infrastructures.**

***Index Terms*—Intrusion Detection, Deep Learning, Early Exits, Energy-efficiency, Edge Computing.**

I. Introduction

THE utilization of resource-constrained devices, particularly in the context of Internet of Things (IoT), has consistently risen over the past years [1]. These devices typically consist of battery-powered embedded computing systems with limited processing capabilities that often include network connectivity, a feature that makes them a common target for attackers. For example, according to a security report [2], 2023 alone experienced a 60% increase in intrusion campaigns compared to the previous year, highlighting the exposure of all connected devices to cyberattacks.

To address this ever-increasing number of network threats, operators often resort to Network Intrusion Detection Systems (NIDSs), implemented through either *rule-based* or *behavior-based* schemes [3]. On the one hand, *rule-based* NIDSs conduct their detection according to a database of previously known threats. Thus, they fall short in detecting new attack variants [4]. On the other hand, *behavior-based* NIDSs assesses the behavior of events and identifies anomalies based on deviations from a previously established normal baseline. As a result, they can identify new types of attacks, provided these attacks exhibit significant deviations from the previously established normal baseline [5].

In recent years, numerous studies have introduced highly accurate *behavior-based* NIDSs, with Deep Neural Network (DNN)-based approaches emerging as the most effective in achieving the highest accuracy levels [6]. To achieve this goal, DNN-based intrusion detection involves forwarding input parameters throughout the entire network architecture until the output layer is reached. In this process, multiple spatial non-linear patterns and features are extracted, subsequently serving as indicators for the classification task conducted at the output layer [7]. To adequately depict network traffic behavior, researchers often increase the complexity of the DNN by adding more parameters and layers [8]. Hence, the accuracy benefits often come at the expense of higher memory and processing requirements, which hinders their application on resource-constrained devices. Surprisingly, the literature often neglects the processing limitations of these devices, frequently assuming that the processing footprint and energy requirements of the employed DNNs can be scaled without constraint [9]. This situation often makes the implementation of DNN-based intrusion detection on IoT devices impractical.

Edge Computing (EC) have emerged as a solution to this challenge in the literature, where the edge, or an intermediate infrastructure device, may cooperate with the cloud to conduct the processing task [10]. More specifically, intrusion detection can be achieved by offloading the DNN-based classification task from the edge device to the edge infrastructure or the cloud. Unfortunately, offloading all network traffic for classification purposes is not easily achievable. This is because network traffic must be classified with minimal processing delays, ensuring that identified attacks are blocked promptly [11]. Notwithstanding, offloading the network classification task can quickly deplete the edge device's bandwidth if all traffic is continuously forwarded to the cloud. In practical terms, edge devices must reliably and autonomously identify which

Manuscript received X XXXXXX 2025; revised XX XXXXXXXX 2024. Date of publication XX XXXXXXX 2024; date of current version XX XXXXXXXX XXXX. This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants nº 304990/2021-3, 407879/2023-4, and 302937/2023-4. (Corresponding author: Eduardo K. Viegas.)

Joao A. Simioni, Eduardo K. Viegas, and Altair O. Santin are with the Pontificia Universidade Catolica do Parana (PUCPR) at the Graduate Program in Computer Science (PPGIa), Paraná Brazil (e-mail: {joao.simioni, eduardo.viegas, santin}@ppgia.pucpr.br)

Everton de Matos is with the Secure Systems Research Center, Technology Innovation Institute (TII), Abu Dhabi (e-mail: everton.dematos@tii.ae)

network traffic needs to be further assessed in the cloud, a task not easily achieved [12]. Adequate NIDS task offloading can improve the overall IoT system's energy efficiency, ultimately paving the way for a green EC, as both edge and cloud infrastructure can benefit from lower resource usage. However, achieving the optimal balance between energy efficiency from offloading and the error rate from local classification remains a significant challenge in the literature.

Combining edge and cloud computing capabilities for intrusion detection is still in its infancy in the literature [13]. A promising approach relies on applying early exits with the deployed DNN architectures. Early exits add multiple termination points that split the neural network into branches, enabling the inference task to conclude prematurely if the currently extracted patterns and features can reliably lead to a decision [14]. Alternatively, the task can be offloaded to the cloud infrastructure when the additional processing capabilities required by the final DNN branches become necessary. While early exits typically reduce the DNN inference computational cost with minimal impact on accuracy, they are not readily applicable to network intrusion detection, especially on resource-constrained devices.

Unlike other domains, network traffic behavior is notably dynamic and continually evolving as time passes [15]. This necessitates adequate DNN model generalization capabilities. Conversely, existing intrusion detection strategies based on early exits often fail to consider the trade-offs in the model generalization that arise from prematurely terminating the inference task at the edge device. Additionally, due to the dynamic nature of network traffic behavior, the classification unreliability caused by an outdated DNN model with a fixed configuration is often assumed to be easily resolved through periodic model updates [16]. A procedure that frequently requires an extended time frame, often days or weeks, and large amounts of labeled training data. As a result, the deployed model must be able to provide a prolonged lifespan to ensure its reliability while an updated model is still unavailable. Under these assumptions, integrating edge and cloud computing for NIDS remains an open challenge in the literature. The offloading strategy must ensure the edge device's security in the presence of non-stationary network traffic behavior and simultaneously support an energy-efficient EC architecture. Achieving this balance is particularly challenging due to the resource-constrained nature of edge devices. The critical challenge lies in determining which set of events requires further analysis with additional computational resources—a decision that must be made autonomously, without operator supervision.

Contribution. In light of this, this paper proposes a new DNN-based NIDS through early exits that operate following an energy-efficient EC architecture, implemented threefold. First, we conduct DNN-based NIDS through early exits coped with a multi-objective optimization strategy for adequate classification offloading. Our model implements the classification task at the edge and proactively offloads events to the cloud when additional processing capabilities are required. Our insight is to leverage multi-objective optimization to find the optimal compromise between accuracy and energy efficiency due to cloud offloading. Secondly, to ensure reliability with new network traffic behavior, we conduct classification with a reject option at the last DNN branch while also calibrating the model's confidence values. The rejection ensures that only highly confident classifications are accepted by our model, while the calibration improves the model generalization. Thirdly, we implement our proposed approach within an energy-efficient EC architecture encompassing edge devices and cloud infrastructure. Our scheme enables reliable NIDS classification offloading from edge devices to cloud infrastructure while optimizing energy efficiency and classification latency.

In summary, the main contributions of this paper are:

- An evaluation of widely used DNN-based NIDS concerning their accuracy, energy consumption and processing costs. Our experiments reveal that current approaches demand unfeasible amounts of processing and energy while also experiencing a rapid degradation in accuracy over time;
- A new DNN-based NIDS with early exits that operates within a energy-efficient architecture. Our proposed scheme can autonomously offload computation to cloud infrastructure when required while also dealing with changes in network traffic behavior;
- A proposal prototype that demonstrates our scheme feasibility under a variety of energy-efficient EC deployment settings. Our scheme can reduce the edge device's energy consumption and processing costs up to only 1%, while keeping or improving F1-Score by 0.02. This is achieved while demanding the offloading of only 10% of network events to the cloud, leading to resource optimization on both the edge and cloud infrastructures;

Roadmap. The remainder of this paper is organized as follows. Section II introduces the fundamentals behind DNN-based NIDS and early exits. Section III describes the related works on EC for *behavior-based* NIDS. Section IV evaluates widely used DNN-based NIDSs with respect to their implementation feasibility on resource-constrained edge devices. Section V introduces our proposed energy-efficient EC architecture for reliable DNN-based NIDS, Section VI describes the prototype implementation, and Section VII evaluates its performance. Finally, Section VIII concludes our work.

II. Preliminaries

This section further describes the operation of DNN-based NIDS and discusses the challenges related to implementing an energy-efficient EC infrastructure in this context.

A. DNN-based Network Intrusion Detection

Network intrusion detection is typically implemented through four sequential modules, namely *Data Acquisition*, *Feature Extraction*, *Classification*, and *Alert* [3]. The *Data Acquisition* module is responsible for the real-time collection of network packets from a monitored Network Interface Card (NIC). These collected packets are then continuously forwarded to the *Feature Extraction* module, which extracts flow-based features to depict network traffic behavior. In general, network traffic behavior is characterized using flow-based

TABLE I: Features set extracted at the network level for each feature grouping in a time window interval of 15s.

Grouping Features	Extracted Features
SRC IP Addresses, SRC/DST IP Addresses, SRC/DST Service Ports	Number of Packets
	Average Packet Size (Bytes)
	Mean Packet Size (Bytes)
	Minimum Packet Size (Bytes)
	Maximum Packet Size (Bytes)
	3rd quartile Packet Size (Bytes)
	Number of Packets (SYN Flag)
	Number of Packets (ACK Flag)
	Number of Packets (ACK + PUSH Flag)
	Number of Packets (FIN Flag)
	Number of Packets (URG Flag)
	Packet inter-arrival time (Avg, Min, Max, Med)

features, which summarize communication patterns between hosts and their services within a defined time window. Table I lists a set of flow-based features typically extracted for NIDS classification tasks. The resulting feature set serves as input to the *Classification* module, which can utilize a DNN model to classify the input features as either *normal* or *attack*. Signaled events are reported by the *Alert* module.

In recent years, extensive research effort has been concentrated on enhancing the detection accuracy of NIDSs, with DNN-based approaches providing the most promising results [6]. DNN-based intrusion detection involves forwarding input parameters through the entire neural network architecture until the output layer is reached. During this process, spatial non-linear patterns and features are extracted and used as indicators for the classification task, which occurs at the final neural network layer [8]. To improve classification accuracy, researchers often increase the number of neural network parameters. Consequently, these methods typically enhance system accuracy, but they come with significant trade-offs regarding memory and processing requirements, which can hinder their application to resource-constrained edge devices.

Notwithstanding, the dynamic behavior of network traffic presents a significant challenge to the fixed configuration of traditional DNN-based NIDSs [16]. Network traffic patterns are not stationary; in contrast, they evolve over time due to various factors such as changing user behaviors, varying application demands, and network configurations. As a result, intrusion detection models trained on static traffic datasets often struggle to adapt to these evolving patterns, leading to a degradation in performance. Fixed DNN-based NIDS configurations, which rely solely on pre-trained models without accommodating these dynamic changes, are particularly vulnerable. The model's accuracy declines as the network traffic evolves, resulting in higher error rates and reduced reliability. To maintain detection effectiveness, periodic model updates are required to adapt to the changing traffic behavior, which is not always feasible in real-world deployments.

B. Early Exits

Early exits are designed to address inference computational cost challenges by introducing side branches that allow for the premature termination of neural network inference tasks [14]. Each side branch typically includes fully connected layers that classify the input at that stage. Suppose an input sample achieves high confidence at a branch. In that case, it can exit from that branch without traversing the remaining neural network layers, reducing the depth some samples need to traverse. As a result, the neural network branches can be distributed, with the first branch executed at the resource-constrained device and intermediate branches offloaded to the cloud infrastructure.

Several approaches can be employed to design an early exit strategy, including modifying the location and structure of side branches within the DNN, as well as customizing the training loss function to optimize the performance of these early exits [17]. Typically, the side branch structure includes a fully connected layer that flattens the output of the preceding layer to perform the classification task. While this approach enables classification at intermediate branches with minimal computational trade-offs, it may struggle if the preceding layer does not provide sufficient features for accurate classification. As a result, the placement of side branches is generally guided by the architecture of the DNN, with particular attention to the location of convolutional layers. Placing a side branch on earlier layers can significantly reduce computational costs. However, it often results in lower accuracies at those branches due to the limited feature representation available at early stages of the network.

Researchers often adopt a joint training approach to conduct the DNN training procedure with early exits [18]. Let N be the number of exit branches, and $\tilde{y}^i$ be the classification output of each branch i on a given event with label y^i. We can compute the joint loss function as a weighted sum of losses of each branch through Equation 1.

$$\mathcal{L}_{joint}(\tilde{y}^i, y^i) = \sum_{i=1}^{N} w_i \mathcal{L}(\tilde{y}^i, y^i) \tag{1}$$

where $\mathcal{L}_{joint}$ is the joint loss function, $\mathcal{L}$ the loss function, and w_i the branch i weight. Here, w_i can be used to fine-tune the accuracies at each branch, such as allowing for a preference towards more accurate earlier branches, resulting in more events classified at the resource-constrained device.

During the test phase, the network inference task processes the input event until it reaches the first branch, potentially stopping the inference prematurely based on whether the classification confidence surpasses a given threshold t. This acceptance threshold is typically set based on the operator's judgment, considering the desired trade-off between accuracy and average inference time. If the final branch is reached, the event's class is determined based on the decision made at that branch.

C. Edge-Computing for NIDS

Edge computing (EC) aims to enable resource-constrained devices, such as those from IoT, to offload their tasks to edge-cloud infrastructure, paving the way for more processing resources [10]. In this context, a energy-efficient EC should address the energy management challenges created by task offloading, such as additional network usage, task dispatch,

and adequate management [19]. Offloading security-related tasks can greatly benefit from EC, given the limited processing capabilities of targeted devices that attackers can explore to circumvent detection [20]. A typical security application involves network traffic analysis for intrusion detection purposes. However, forwarding all network traffic for offloading the analysis task is not practical in EC settings, as it can quickly saturate the device's network bandwidth. Therefore, approaches capable of effectively determining when network traffic requires additional processing resources are essential.

Splitting NIDS-related tasks from resource-constrained devices to edge-cloud infrastructures is still a challenge in the literature [21]. Although early exits for DNNs have shown great success in several tasks, their application for NIDS in EC is still in its infancy. Finding the optimal compromise between accuracy, energy efficiency, and classification latency poses a significant challenge to designed schemes. This is because offloading a neural network branch from edge to cloud (see Section II-B) also requires sending the features extracted by the prior branch. As a result, the benefits of energy efficiency and accuracy can often only be achieved with significant tradeoffs in network communication.

Another challenge arises from the dynamic behavior of network traffic, which can be caused by the discovery of new attacks or even the provision of new services [15]. Changes in network traffic behavior cause designed DNN-based schemes to degrade in accuracy compared to their performance measured during the testing phase [16]. Current approaches assume that periodic model updates are conducted to adequately account for these changes, often requiring weeks or even months to complete. Conversely, new network traffic must be collected and its behavior adequately labeled as *normal* or *attack*, often requiring human assistance. Only then can the computationally expensive process of model training be conducted. This situation often leads to the unreliable use of outdated DNN models in production for extended periods before an updated model becomes available.

Under these circumstances, splitting NIDS-related tasks between resource-constrained devices and cloud infrastructure can pose a significant challenge, as the deployed model may generate a higher rate of false alerts than anticipated. This situation highlights the need for the DNN model to adequately generalize to changes in network traffic behavior over time. As a result, achieving reliable offloading of DNN-based NIDS while considering a EC setting remains a challenge in the literature.

III. Related Works

This section further assesses the current landscape of DNN-based NIDS in EC infrastructures. Specifically, we first examine DNN-based approaches tailored for resource-constrained devices, followed by an investigation into how EC facilitates reliable NIDS offloading tasks.

A. Behavior-based NIDS for Resource-constrained Devices

In general, DNN-based NIDSs proposed in the literature aims at improving detection accuracy on specific datasets [22]. For example, M. Ge *et al.* [23] propose a DNN-based approach for intrusion detection in IoT. Their scheme utilizes embedding layers for high-dimensional categorical feature encoding, which improves classification accuracy. However, they assume a static behavior of network traffic while overlooking the associated processing costs. Similarly, V. Ravi *et al.* [24] employ a feature dimensionality reduction technique in their DNN model to reduce processing requirements. Their model outperforms related works but overlooks changes in network traffic behavior. H. Nandanwar *et al.* [25] propose a DNN-based approach for IoT attack detection using a Gated Recurrent Unit (GRU) for temporal feature representation. Their approach improves classification accuracy but overlooks the computational costs associated with applying GRU on resource-constrained devices.

In recent years, several authors have pursued more resource-efficient approaches for intrusion detection, aiming to facilitate their implementation on IoT devices. For instance, S. Khandelwal *et al.* [26] reduce the precision of floating point operations in their DNN model to 2 bits. Their scheme significantly decreases inference costs with minimal impact on accuracy. However, the model's performance in assessing highly dynamic network traffic behavior is overlooked. Another approach based on model quantization was proposed by L. Zhang *et al.* [27], where a DNN model is converted to a binarized version with no significant impact on accuracy. Their scheme significantly improves inference throughput but overlooks the impact on model generalization for network traffic analysis. Reducing the number of input features is another effective method to minimize processing costs while maintaining detection performance. R. Zhao *et al.* [28] implement a feature reduction approach to decrease the number of required DNN parameters. While their scheme improves inference throughput, it overlooks the implications of model generalization and how it affects reliability. Similarly, O. R. Sanchez *et al.* [29] propose a feature selection technique to decrease inference costs while maintaining classification accuracy. Although their model significantly reduces processing costs, it also neglects the implications on model generalization capabilities.

Intrusion detection under highly dynamic network traffic behavior remains a significant challenge in the literature [4]. Generally, authors assume that model updates can be easily conducted to address this issue, often overlooking the associated challenges (see Section II-C). As an example, O. A. Wahab [30] employs a drift detector combined with a feature reduction scheme to identify changes in network traffic behavior on IoT devices. Their model aims to maintain stable accuracy performance over time but assumes a supervised learning setting where a network operator can assess the model's accuracy and label events accordingly. Another approach was proposed by G. Andresini *et al.* [31], where a semi-supervised approach detects concept drifts. The authors incrementally adjust the deployed model using events with new behaviors in an active learning strategy. However, they only evaluate their performance on synthetic data and overlook resource-constrained device applications.

B. Edge Computing Offloading

Several authors have increasingly adopted EC strategies in recent years, reporting promising results to facilitate access to additional processing resources [21]. In such contexts, proposed approaches vary from enhancing intrusion detection accuracy to effectively managing task offloading. For instance, X. Zhao *et al.* [32] propose a reinforcement learning strategy to select an edge server for offloading intrusion detection tasks based on energy consumption and detection delay metrics. Their approach aims to reduce IoT device energy consumption but overlook trade-offs in accuracy and network traffic usage. Another approach to task offloading through blockchain is presented by H. Liu *et al.* [33]. Their method aims to minimize processing trade-offs through blockchain-based offloading but neglects network communication and energy efficiency. A. Mourad *et al.* [34] propose offloading intrusion detection tasks based on the proximity of edge devices. While their model reduces resource usage on IoT devices, it overlooks challenges such as the dynamic behavior of network traffic and the communication overheads associated with intrusion detection.

In recent years, a popular approach to task offloading has involved the use of DNN inference splitting strategies (see Section II-B). For instance, M. Ayyat *et al.* [35] proposed the class-aware early exit DNN to prioritize class importance at different side branches. Their approach reduces inference processing costs while enhancing classification accuracy on resource-constrained devices. However, the authors did not implement the DNN side branches in a distributed manner. A. Bakhtiarnia *et al.* [36] introduced a new training strategy for early exits where earlier branches are optimized to classify more samples. This approach enhances inference throughput but does not address deployment in a distributed setting. R. G. Pacheco *et al.* [37] distributed DNN side branches in an edge-cloud environment for effective task offloading. They highlighted the critical role of communication bandwidth in inference offloading and examined the impact of cloud location on Round Trip Time (RTT) time. However, their evaluation was limited to vision-related tasks. In a subsequent work [14], they emphasized the significant impact of classification confidence calibration on effective task offloading. As a result, applying early exits within the NIDS domain remains an open challenge in the literature. This is primarily due to the complexity of achieving an optimal trade-off between energy consumption and error rate during task offloading. Determining when to terminate inference locally or offload it to the cloud requires careful calibration, as premature offloading may lead to higher energy consumption, while incorrect local classification can increase error rates. This balancing is further complicated by the dynamic nature of network traffic, which necessitates adequate mechanisms to maintain reliability in real-time intrusion detection scenarios.

C. Discussion

Current approaches to DNN-based NIDS for resource-constrained devices typically focus on reducing the computational cost of the inference task, employing techniques such as model quantization [26], [27] and feature reduction [28], [29]. While these methods show promise in enhancing efficiency, they often fall short of accounting for the trade-offs in model generalization, particularly in dynamic network traffic environments [16]. This inability to adapt to the evolving nature of network traffic is a significant limitation, as these methods may struggle to maintain accuracy in real-world, ever-changing conditions. Additionally, while offloading tasks to edge or cloud infrastructures has been proposed as a means to overcome hardware constraints, the literature often overlooks the critical trade-offs between energy efficiency and reliability in NIDS-related tasks [32], [33]. In this context, DNN splitting strategies may effectively facilitate intrusion detection offloading when necessary, but authors frequently overlook their implementation in a distributed and realistic manner [35], [36].

In response to these challenges, our proposal addresses several key gaps in the literature. Specifically, we focus on enabling the deployment of DNN-based NIDSs on resource-constrained devices by improving both energy efficiency and reliability in dynamic network traffic environments. We aim to enhance the reliability of intrusion detection systems in scenarios where network behavior continuously evolves—an issue that has been largely overlooked in current solutions. Furthermore, we tackle the research gap in existing methods by optimizing the balance between energy efficiency and reliability during intrusion detection offloading. Additionally, we explore a DNN splitting strategy for distributed offloading. This topic has been overlooked in the literature but is a must for practical, and scalable deployment of DNN-based NIDSs in real-world settings.

TABLE II: *MAWIFlow* dataset statistics.

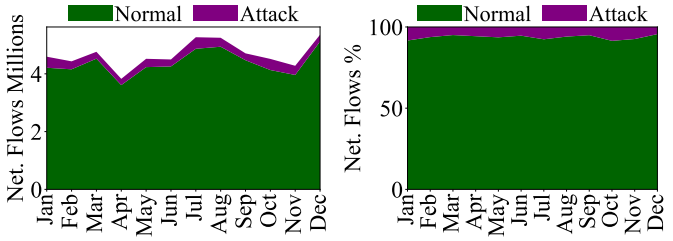
Property	Value
Average Daily Network Packets	105 Millions
Average Daily Network Flows	9 Millions
Average Daily Throughput	610 Mbps
Average Daily Anomalous Flows	1.8 Millions
Average Daily Dataset Size	19.7 GB
Total Network Packets	27.72 Billions
Total Network Flows	6.14 Billions
Total Dataset Size	7.1 TB

IV. Problem Statement

In real networked environments, deployed DNN-based NIDSs must accurately capture the non-stationary behavior of network traffic. In this section, we delve deeper into the performance of widely used DNN-based NIDSs in detecting network intrusions on resource-constrained devices. We begin by introducing the dataset utilized in our study. Subsequently, we evaluate the performance of various DNN models in terms of accuracy and processing costs.

A. A realistic network intrusion dataset

Current datasets widely used in the literature often assume a static behavior of network traffic [38]. These datasets are typically divided into training and testing sets, neglecting the dynamic and evolving nature of real-world network traffic.



(a) Number of network flows. (b) Distribution of network flows.

Fig. 1: *MAWIFlow* network flow distribution over the year.

Consequently, designed schemes can typically achieve high detection accuracies during the testing phase but may perform poorly when deployed in production. This scenario becomes increasingly challenging over time as new services are introduced or novel attack patterns are discovered.

To address such a challenge, we utilize the *MAWIFlow* dataset, a publicly available intrusion dataset containing real, valid, and labeled network traffic collected from production environments spanning an extended period. To achieve these characteristics, the *MAWIFlow* dataset is built upon the MAWI [39] working group traffic archive. It includes network traffic from MAWI samplepoint-F, a transit link between Japan and the USA, collected in 15-minute intervals. The network data is collected daily, resulting in a network PCAP file for each day over the evaluation period, totaling over 7TB of data and encompassing more than 70 billion in network flows. For this research, we utilized the network data collected throughout the entire year of 2016. Table II presents the statistics of the *MAWIFlow* dataset, while Figure 1 illustrates the distribution of network flows over the year.

The collected data is organized into network flows based on the hosts and services involved in each communication. Each network flow represents a 15-second segment of client/service and server/service data, which is then summarized into an associated feature set. For this study, we extracted 58 features from Moore's work [40]. Table I lists the set of features used. To assign labels, our study employs the MAWILab [41] unsupervised machine learning algorithms designed to identify network anomalies.

Consequently, the constructed dataset facilitates the evaluation of proposed intrusion detection schemes, particularly in assessing their accuracy over time. This is because *MAWIFlow* includes daily labeled network traffic data collected from a real network environment over a year-long period. The resulting dataset allows for an in-depth analysis of how the dynamic behavior of network traffic impacts the accuracy of the proposed schemes. In addition, the built dataset is publicly available for the research community.

B. Model Building

To investigate the accuracy degradation and processing costs on resource-constrained devices, we evaluate the performance of two widely used DNN architectures: AlexNet and MobileNetV2. This evaluation covers a resource-demanding architecture, demonstrated by AlexNet, and a more lightweight counterpart, represented by MobileNetV2. The selected DNN architectures are widely used for intrusion detection to enhance the accuracy of designed systems. Therefore, we use them as a traditional baseline approach to evaluate their performance in terms of both accuracy and energy consumption.

Both architectures were adjusted to fit *MAWIFlow* tabular format (features are listed on Table I). To be more precise, we first transform the 58 features input into a single-channel 8x8 square shape, zeroing the remaining features during such a process. This adapted input is then reshaped to a 48x48x1 using average pooling, subsequently serving as input to the DNN architectures. In practice, the resulting feature conversion is represented as a one-color image, which can be used as input by the DNNs. The goal is to enable the appropriate transformation of the input format from MAWIFlow into a matrix format required by the selected DNNs, ensuring that no information is lost during the process. This is achieved by applying average pooling to the generated matrix and leveraging the convolutional layers of the selected DNN architectures to extract and preserve the feature relationships.

The DNNs were trained using *adam* optimizer, running for $1,000$ training epochs, with a batch size of $1,000$ events. We utilized *categorical cross-entropy* as the loss function. The learning rate was set at 0.001 with a learning rate scheduler that stops training if there is no improvement in the validation accuracy over 50 epochs. To ensure the proper model training, as most events on the MAWIFlow dataset are normal (see Table 1), we apply a random undersampling without replacement at every training task. Similarly, we also apply a *min-max* scaling procedure to adequately normalize the dataset between -1 and $+1$. These models were implemented using PyTorch API version 2.1.0.

We evaluate the selected classifiers using the following classification performance metrics:

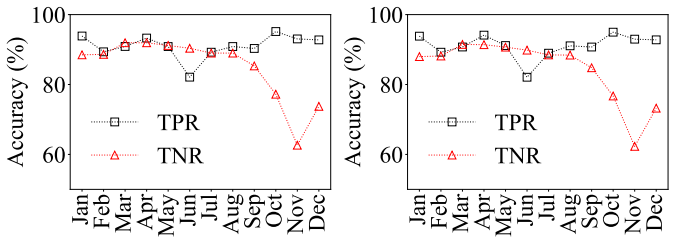
- *True Positive (TP)*: number of attack samples correctly classified as an attack.
- *True Negative (TN)*: number of normal samples correctly classified as normal.
- *False Positive (FP)*: number of normal samples incorrectly classified as an attack.
- *False Negative (FN)*: number of attack samples incorrectly classified as normal.

Further, we measure the F-Measure according to the harmonic mean of precision and recall values while considering attack samples as positive and normal samples as negative, as shown in Eq. 4.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F\text{-}Measure = 2 \times \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$



(a) AlexNet (b) MobileNetV2

Fig. 2: Accuracy trends of commonly used classifiers over a year without periodic model updates. The classifiers are trained in January and evaluated in the subsequent months without updates.

C. DNN-based Network Intrusion Detection for Resource-Constrained Devices

This section further investigates the accuracy performance and processing costs of widely used DNN-based NIDSs. More specifically, we aim to answer the following Research Questions (RQs):

- ***RQ1:*** *What is the accuracy performance of DNN-based NIDS when exposed to non-stationary network traffic?*
- ***RQ2:*** *What are the processing costs of selected techniques on resource-constrained devices?*

Our first experiment aims to address RQ1 by investigating the accuracy performance of the selected DNN architectures for intrusion detection. To achieve this, we train each selected DNN architecture using data from the entire month of January (see Fig. 1) and evaluate the model's performance over the remaining year without periodic updates. This simulates a realistic scenario where the designed schemes must operate for extended periods before an updated model becomes available. The experiment aims to determine if the current widely used DNN-based NIDSs in the literature can reliably perform intrusion detection under non-stationary network traffic behavior.

Figure 2 shows the monthly accuracy performance of the selected techniques without periodic model updates. The evaluation indicates that these techniques provide significantly high detection accuracies in the initial months following the training period. However, as time progresses, there is a noticeable downward trend in accuracy, as indicated by the decline in TN rates towards December. For instance, the selected DNNs showed a decrease in TN rates in November compared to their accuracies in January by 28% and 29% for AlexNet and MobileNetV2, respectively. This trend poses a significant challenge for intrusion detection, especially for resource-constrained devices, because of the delay required in updating the deployed DNN models. Consequently, this evaluation demonstrates that current DNN-based approaches in the literature cannot reliably perform intrusion detection under non-stationary network traffic behavior. Furthermore, relying solely on a computationally demanding DNN for reliability improvement, as measured by AlexNet, does not yield a significant accuracy improvement compared to its lightweight counterpart, MobileNetV2.

Our second experiment aims to answer RQ2 by evaluating the processing costs of the selected DNN techniques for implementation on resource-constrained devices. We assess the average inference computational costs in both Desktop and Raspberry Pi environments. The Desktop environment is equipped with a 16-core Intel Xeon E5-2640 v3 CPU, 32 GB of memory, and an Nvidia Tesla T4 GPU running on Ubuntu Linux 22.04. The Raspberry environment is a Raspberry Pi 3 Model B, with a 4-core Broadcom CPU and 1 GB of memory running on Raspberry Pi OS with kernel version 6.1. The goal is to investigate further the implementation feasibility of the selected DNN architectures in a resource-constrained setting.

TABLE III: Average event detection throughput (events/sec).

DNN	System		
	Raspberry	Desktop	
		CPU	GPU
AlexNet	7.36	247.34	17,609
MobileNetV2	7.93	509.58	3,419

Table III shows the average event detection throughput, measured by the number of classified events per second. Evidently, the use of a GPU for inference significantly impacts detection throughput, as demonstrated by the Desktop environment utilizing the GPU. However, when deployed on a resource-constrained device, the selected DNNs achieves poor detection throughput, reaching only ≈ 7 events per second for both architectures. Consequently, despite their challenges in providing reliable detection under non-stationary network traffic conditions, their low detection throughput on resource-constrained devices renders them unsuitable for production implementation. Surprisingly, this situation applies to both selected DNN architectures, regardless of their number of parameters.

D. Discussion

This section evaluated the accuracy and processing impact of widely used DNN architectures for intrusion detection. The experiments demonstrated that current approaches could not cope with the evolving behavior of network traffic, leading to a significant degradation in accuracy months after the training period (Fig. 2). Regardless of the complexity of the used DNN model, the evaluated approaches fail to provide reliable detection, highlighting the need for new methods to address the non-stationary nature of network traffic. Additionally, the detection throughput of the evaluated schemes makes them unsuitable for implementation on resource-constrained devices (Table III). Therefore, ensuring detection reliability for evolving network traffic while enabling implementation on resource-constrained devices remains a challenge in the literature.

V. A DNN-based NIDS with Early Exits for Detection Offloading in Edge Computing

To address the aforementioned challenges, our work proposes a new DNN-based NIDS through early exits that operate following an energy-efficient EC rationale. The workflow of our proposed scheme is illustrated in Figure 3, and is implemented in three main stages.

First, we conduct intrusion detection using a DNN model with early exits. Our insight is to leverage early exits to dis-

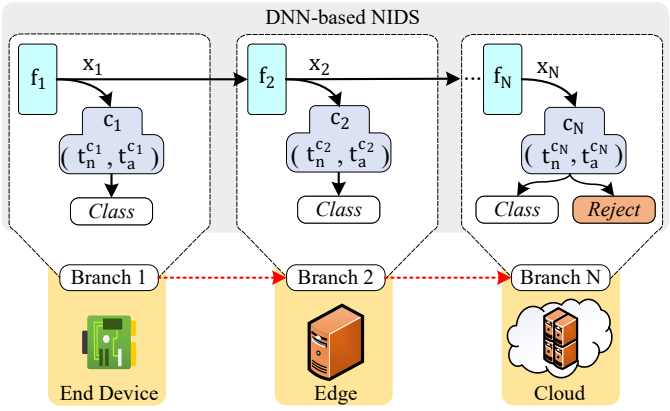


Fig. 3: Workflow of the proposed DNN-based NIDS through early exits for intrusion detection offloading in EC settings. DNN Branches are distributed from end device, edge, and cloud infrastructure.

tribute the DNN-based inference NIDS task within an energy-efficient EC architecture. Consequently, the intrusion detection task can be split between the *End Device*, *Edge*, and *Cloud* infrastructure based on the classification difficulty of the evaluated event. This approach aims to offload computation only for a subset of events requiring additional processing capabilities, thereby simultaneously decreasing the *End Device*'s energy consumption and reducing the *Cloud* infrastructure's resource usage. Second, to find the optimal tradeoff for each DNN branch (Fig. 3, *Branch* 1 to N), we conduct a multi-objective optimization task. We aim to adequately assess the accuracy and energy consumption tradeoff when offloading intrusion detection tasks among the EC entities. Third, to improve the model's generalization capabilities for adequately addressing the non-stationary behavior of network traffic, we perform the classification task with model calibration and a reject option. On the one hand, model calibration ensures that the model's confidence values accurately reflect their reliability, enabling a reliable assessment of classification confidence. On the other hand, the reject option in the final model branch (Fig. 3, *Branch* N) allows the model to reject potential misclassifications at the *Cloud* infrastructure. As a result, our proposed model can reduce the energy consumption of *End Devices* and the resource usage of the *Cloud* infrastructure, improve the model's generalization capabilities, and ultimately pave the way for an effective energy-efficient EC.

The next subsections further describe the implementation of our proposed model, including the modules that implement it.

A. A NIDS Model With Early Exits

Current DNN-based NIDSs demand computational processing capabilities that are unfeasible for implementation on resource-constrained devices (see Table III). To address this, edge offloading for DNNs can be performed based on the input's complexity by introducing early exits. This is because conducting the side branches on the same *End Device*, as often assumed in the literature, can still pose challenges related to energy consumption and processing requirements for events reaching the final branches. To address these challenges, our model distributes the deployment of DNN branches between *End Device*, *Edge*, and *Cloud* infrastructure according to the current evaluated event, as illustrated in Figure 3.

Consider an intrusion detection task where $x \in \mathbb{R}^D$ denotes a D-dimensional feature vector input and $y \in \{n, a\}$, where n denotes a *normal* event, and a denotes *attack* labeled events. Our model aims at learning a DNN that can model the probabilistic predictive distribution $p(y|x)$ over ground truth labels. Following a typical early exit framework, during the model learning process, we introduce N classifiers $c_{\{0,\dots,N\}}$ for a given set of intermediate DNN layers $f_{\{0,\dots,N\}}$. Namely, let c_i be the i-th classifier that receives the x_i feature vector output by the f_i DNN layer, the DNN will prematurely end inference if the c_i classification confidence level surpasses the classification threshold t^{c_i} for *normal*-classified ($t_n^{c_i}$), or *attack*-classified ($t_a^{c_i}$) events, as illustrated in Figure 3.

Here, the classification threshold t^{c_i}, denotes a tuple where $t_n^{c_i}$ establishes the acceptance threshold for *normal* events, and $t_a^{c_i}$ the acceptance threshold for *attack* events. As a result, relying on lower t^{c_i} for a given classifier c_i can increase the number of accepted events, whereas using higher t^{c_i} will lead to a higher acceptance rate on branch i. The acceptance thresholds should be defined based on the operator's needs, as they directly impact energy consumption and processing costs. The proposed approach for optimizing the acceptance thresholds is further described in Section V-B.

In contrast to traditional early exit strategies, to further increase the model's generalization, we incorporate a reject option at the last DNN branch to reliably handle the classification decision (Fig. 3, *Branch N*). Our model accepts or rejects the classification at the final branch based on its associated confidence value $\hat{p}$. To achieve such a goal, the module's implementation is coped with a rejection rej function, as determined by the following equation:

$$rej(\hat{p}, t^{C_N}) \begin{cases} \emptyset & \text{if } \hat{p} \leq t^{C_N} \\ \hat{p} & \text{otherwise} \end{cases} \tag{5}$$

where $\emptyset$ denotes events likely to be incorrect decisions the DNN model final branch performs, and t^{C_N} the acceptance thresholds at the final model branch. As a result, the rej module suppresses unreliable classification as measured by their associated classification confidence values at the final DNN branch (Fig. 3, *Branch N*). Similarly, the rejection threshold should be determined based on the operator's judgment. A higher rejection threshold will enhance system reliability but result in a higher proportion of rejected events. Conversely, a lower threshold will accept more events but expose the system to unreliable classifications.

Therefore, by relying on our proposed early exit approach, the NIDS inference task can be split between the *End Device*, *Edge*, and *Cloud*. When offloading a given branch i, the network tradeoff only involves sending the output of the current DNN layer f_i, namely vector x_i, as opposed to the traditional approach where the entire network traffic must be transmitted. Thus, our proposed model enables the adequate NIDS inference splitting in a EC setting.

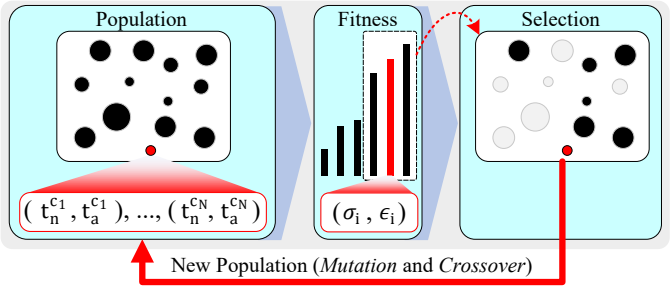


Fig. 4: Multi-objective optimization for DNN-based NIDS with Early Exits. Here, σ_i measures the average DNN inference energy consumption, and ϵ_i measures the error rate obtained using the selected acceptance thresholds t^{c_i}.

B. Multi-objective Optimization

Finding the optimal acceptance thresholds $t^{c\{1,\dots,N\}}$ for each branch i is challenging. This is because each branch threshold affects the subsequent branches' performance while directly influencing the overall energy consumption and accuracy. Given this challenge, we conduct the model building as a multi-objective optimization task, as illustrated in Figure 4.

We consider a DNN model h implemented with N branches, coped with a *rej* module at the final branch N (see Section V-A). In such a case, the goal of the multi-objective optimization is to find N associated $t^{c\{1,\dots,N\}}$ branches thresholds that simultaneously minimize the resulting system energy consumption (σ) and the error rate (ϵ). Our central assumption is that energy consumption decreases with the acceptance of additional events at earlier branches. Conversely, the error rate decreases with a higher rejection rate, which leads to increased processing costs as more events are routed to the final branch.

Therefore, we can conduct the multi-objective by solving the following equation:

$$\begin{aligned} &\underset{\{t^{c_1},\dots,t^{c_N}\}}{\arg\min}\ \sigma(h(\mathcal{D},\{t^{c_1},\dots,t^{c_N}\})) \\ &\text{and} \\ &\underset{\{t^{c_1},\dots,t^{c_N}\}}{\arg\min}\ \epsilon(h(\mathcal{D},\{t^{c_1},\dots,t^{c_N}\})) \end{aligned} \tag{6}$$

where h denotes the DNN model with multiple branches, coped with our *rej* (see Fig. 3). Here, σ is a function that measures the model average inference energy consumption on a given dataset $\mathcal{D}$ when using the $\{t^{c_1},\dots,t^{c_N}\}$ thresholds, whereas ϵ is a function that measures the resulting error rate. As a result, our proposed scheme aims to identify the optimal system thresholds that simultaneously minimize energy consumption and error rates. In practice, we can solve equation 6 by implementing it as a multi-objective optimization, later discussed in Section VII-C, our energy consumption measurement approach is presented in Section VII-B.

Therefore, in terms of overheads, the computational requirements for solving the multi-objective optimization process are only necessary during the training phase, which occurs off the end device. During this phase, the optimization process is performed to compute the trade-offs between error rate and energy consumption for the classification branch thresholds. Once trained, the system operates with the predefined thresholds. Additionally, to address dynamic network environments, the network operator has the flexibility to periodically reevaluate and adjust the classification thresholds without requiring model updates. This allows for the rejection of a greater number of events in response to changing conditions, thus giving the network operator more time to perform model training tasks as needed.

C. Addressing the Generalization Challenge

The network traffic behavior is highly dynamic and changes over time. This non-stationary behavior poses a challenge for current DNN-based NIDSs, as they often struggle to generalize network traffic behavior adequately (see Fig. 2). In this context, a key assumption of our model involving early exits revolves around assessing the classification confidence of the model's branches to terminate inference prematurely. Therefore, the model's predicted confidence vector should reflect the ground-truth probabilities of the correctness of the model.

However, DNNs are known to produce both overconfident and underconfident classifications for input events. For example, if the model predicts a probability of 0.7 for a given class, it is expected that out of 100 predictions, approximately 70% of those are correct. If the percentage of correct predictions is below 70%, the model is overconfident, whereas if it is above 70%, it is underconfident. Therefore, to reliably deploy DNN-based NIDSs with early exits, it is essential to ensure that their confidence levels accurately reflect their expected accuracy.

To address such a challenge, our proposed scheme conducts the confidence calibration of our designed DNN model (see Section V-A). In practice, we calibrate each DNN branch confidence output to ensure their sampled confidence can function as their expected accuracy. Given this goal and recognizing that achieving perfect model calibration is not feasible because the model confidence is a continuous random variable, we conduct model calibration through empirical approximation.

We aim to calibrate each classifier c_i at every branch i, such that their output classification confidence $\hat{p}$ reflects their accuracy. To estimate the expected accuracy from a finite number of samples with a continuous random variable, we group predictions into M interval bins, each of size $1/M$, and compute the accuracy over each bin. Each bin separates the classification confidence into finite intervals. For example, assuming $M = 10$, the first bin will contain all events with classification confidence values ranging from 0.0 to 0.1, whereas the second bin will contain those ranging from 0.1 up to 0.2.

Let B_m be the samples whose classification confidence $\hat{p}$ falls within the bin interval m, the accuracy of B_m can be computed as:

$$\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} (\hat{y}_i = y_i) \tag{7}$$

where $\hat{y}_i$ is the predicted label, and y_i the true label. Then, we can compute the average confidence within the bin interval B_m according to the following equation:

$$\text{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i \tag{8}$$

where $\hat{p}_i$ is the classifier confidence for event i. Thus, a perfectly calibrated model will have $\text{acc}(B_m) = \text{conf}(B_m)$ for every $m \in \{1, ..., M\}$. Finally, we can measure the calibration correctness through Expected Calibration Error (ECE) approach, as follows

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)| \tag{9}$$

where n is the number of samples. The difference between acc (Eq. 7) and conf (Eq. 8) denotes the calibration *gap* for a given bin. The ECE can be used as an empirical metric of model calibration, where values close to zero denote a perfectly calibrated model.

Our proposal uses the ECE metric to calibrate the post-training model. To achieve this, we rely on the temperature scaling technique. Each DNN branch classifier outputs a vector called logits, which is then passed through a *softmax* function to obtain the class probabilities. The temperature scaling goal is to find a temperature vector T that can be used to divide the DNN logits such that it minimizes the resulting model ECE. Hence, we can use temperature vector T during the inference phase as follows:

$$\text{softmax}(z_i) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \tag{10}$$

where z_i is the sample i logit, and T the temperature vector. Temperature scaling does not affect the resulting model's accuracy, as T does not change the maximum of the *softmax* function. Therefore, it is possible to conduct simple parameter search to find a T vector that can reduce ECE. The resulting model will then use T to calibrate the classification confidence for later use to measure classification correctness.

D. Discussion

Our proposed model aims to facilitate the offloading of DNN-based NIDS inference while complying with energy-efficient EC requirements. To achieve this goal, we rely on the application of early exits, where model branches can be deployed across *End Device*, *Edge*, and *Cloud* infrastructures (see Fig. 3). To ensure our model can reliably handle the dynamic behavior of network traffic, we incorporate a reject option at the final model branch executed in the cloud, which aims at identifying unreliable decisions over time (see Section V-A). In addition, we formulate the threshold finding as a multi-objective optimization task to determine the optimal tradeoffs between energy consumption and error rate (see Eq.6). Finally, we calibrate the resulting model confidence values to effectively generalize network traffic behavior while employing early exits, ensuring a reliable indication of model correctness (see Eq.10). As a result, our proposed model facilitates the implementation of DNN-based NIDS on resource-constrained devices. Our approach enables efficient offloading while decreasing energy consumption at the *End Device* and lowering the resulting system error rate.

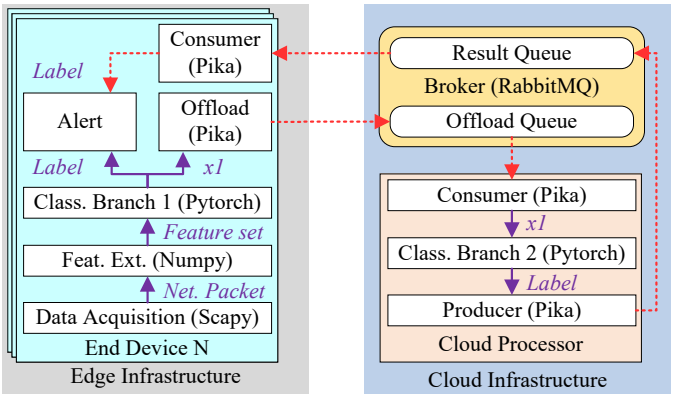


Fig. 5: Prototype implementation overview of our proposed Edge Computing DNN-based NIDS model.

In addition, the multi-objective optimization model allows the network operator to define the desired acceptance rate for the utilized DNN branches. This is because the rejection rate should be carefully determined based on the operator's specific needs. A high rejection rate may lead to a substantial number of events being offloaded to the cloud environment, potentially increasing costs associated with storage and processing, such as for later model update purposes. Conversely, accepting more samples on the first DNN branch at the end device could result in a higher error rate. In practice, our model provides flexibility for the network operator to adjust the operation points dynamically based on current network conditions. This approach allows for gradually increasing rejection rates as the deployed model ages, ensuring system reliability until the model update process is performed.

VI. Prototype

We implemented a proposal prototype in a distributed environment as illustrated in Figure 5. It considers the implementation of multiple *End Devices* executed on a *Edge Infrastructure* and a single *Cloud Infrastructure* for executing the offloaded task. The prototype's goal is to evaluate the performance of our proposed mechanism. To achieve this, the hardware and software components of the prototype are designed to create an edge computing environment where end devices operate under resource constraints. At the same time, the cloud infrastructure provides additional computational capabilities when required. The selected network protocols are intended to facilitate a lightweight communication interface between the end devices and the cloud infrastructure, ensuring low communication overhead within edge computing architecture.

Each *End Device* executes our DNN-based NIDS pipeline with our proposed model (see Fig. 3). To achieve this goal, it continuously collects the network packets from a given monitored NIC through Scapy API $v.2.5.0$. The behavior of the collected packets is then extracted using numpy API $v.1.26$, compounding a feature set with the features listed on Table I. We implement a single early exit on the chosen DNNs, where the first branch is executed at the *End Device*, and the

second branch at the *Cloud Infrastructure*. The details regarding the DNN early exit implementation are later discussed in Section VII-A. The extracted feature set is used as input to the first DNN branch, implemented through Pytorch API $v.2.1.0$ (Fig. 5, *Class. Branch* 1). The associated event label is used as input to the *Alert* module, if its classification confidence level surpasses acceptance threshold (Fig. 3, t^{c_1}). Otherwise, the output of the intermediate DNN layer at the first branch is offloaded to the cloud (Fig. 5, *Offload*).

The communication channel between the *Edge Infrastructure* and the *Cloud Infrastructure* was implemented through a RabbitMQ broker. To this end, the prototype relies on two queues, namely *Offload* and *Result*. The first sends the first branch intermediate output from the *End Device* to the cloud, whereas the latter sends back the associated Label from the second branch as executed at the cloud. The sending and reading of the RabbitMQ messages were implemented through Pika API $v.1.3.2$.

At the *Cloud Infrastructure*, we deploy the RabbitMQ broker and the second DNN branch. To achieve this objective, we execute a *Cloud Processor* module, which continuously consumes the *Offload Queue* messages and feeds them as input to a *Class. Branch 2* module. The module executes the second (final) DNN branch and forwards the resulting Label to the *Producer* module, which produces a message to the *Result* RabbitMQ queue. Finally, the *End Device* reads the generated message through a *Consumer* module and forwards it to the *Alert* module.

To generate a resource-constrained device behavior, we implemented the *End Device* through a Raspberry Pi 3 Model B, with a 4-core Broadcom CPU and 1 GB of memory running on top of Raspberry Pi OS with kernel version 6.1. The *Cloud Processor* module was implemented through a dedicated Virtual Machine (VM) on top of IBM Cloud Computing. The VM was equipped with 2 virtual CPU cores, 8 GB of memory running on top of Ubuntu OS $v.24.04$. We execute the *Cloud Infrastructure* through multiple zones to adequately measure the resulting trade-offs of our scheme (latter discussed in Section VII-D).

VII. Evaluation

The proposal evaluation aims at answering the following RQs:

- ***RQ3:*** *How does our proposed multi-objective optimization improves system performance?*
- ***RQ4:*** *Does our proposed model improves classification reliability?*
- ***RQ5:*** *What are the system tradeoffs when implemented in an EC architecture?*

The subsequent subsections provide further details about the implementation of our model and its performance.

A. Model Building

We evaluate our proposed model using the same DNN architectures evaluated previously (see Section IV). To achieve this goal, we introduce the first set of early exits immediately after the initial features are extracted, ensuring that the computational costs remain low in the first branch. The second branch, on the other hand, executes the remaining DNN layers, extracting additional features for the classification task but requiring more computational resources. The early exit implementation strategy was made based on related works [17]. More specifically, we introduce a single early exit component on each chosen DNN, as follows:

- ***AlexNet.*** A classifier is introduced between the 1st and 2nd convolutional layers. The classifier flattens the 1st convolutional layer output by applying a fully connected layer with $1,600$ input neurons followed by 2 neurons output;
- ***MobileNet.*** A classifier is introduced after the 2nd bottleneck layer. The classifier flattens the preceding layer output, followed by a 0.2 dropout layer and a fully connected layer with $1,280$ input followed by 2 neurons output;

Therefore, each evaluated DNN architecture consists of two branches encompassing the added layers from the first branch, while the last branch comprises the traditional DNN output (Fig. 5, *Class. Branch* 1 and 2). The modified models are trained through the joint loss function (see Eq. 1) with *categorical* loss for each branch, with a 1.0 branch layer weight w, with $1,000$ training epochs. The learning rate was set at 0.001, with a batch size of $1,000$ events. These models were implemented using PyTorch API version $2.1.0$.

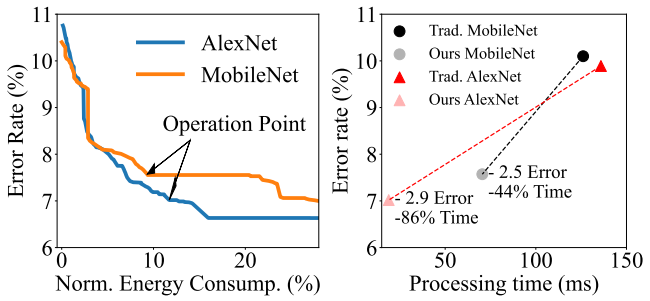
B. Energy Consumption Measurement

To measure the system's energy consumption, we use a current-voltage power ratio consumption meter connected to the electrical outlet that powers the Raspberry Pi 3. In practice, we first measure the idle power consumption and then compute the difference in energy consumption when performing the inference task (using the proposed early exit) on the Raspberry Pi. It is important to note that the inference task energy consumption also includes the offloading to the cloud when the second DNN branch is required (see Fig. 5). Therefore, the measurement replicates a realistic setting where a portion of events are classified at the *End Device* using the 1st branch, while the remaining events are classified using the 2nd branch executed at the *Cloud Infrastructure*.

C. DNN-based NIDS with Early Exits

Our first experiment aims to answer *RQ3* and investigates how our proposed multi-objective optimization can improve the system performance. To achieve this goal, we aim to find the optimal tradeoff between energy consumption and error rate (see Eq. 6) according to the used classification thresholds (t^c). In practice, we adjust the classification thresholds for each selected DNN to account for the application of two early exits. The multi-objective optimization aims at finding the optimal classification thresholds for the first branch at the *End Device* ($t_a^{c_1}$, $t_n^{c_1}$), and the second branch at the *Cloud Infrastructure* ($t_a^{c_2}$, $t_n^{c_2}$).

To achieve this goal, we implement our scheme as a multi-objective optimization task using the *Non-dominated Sorting*



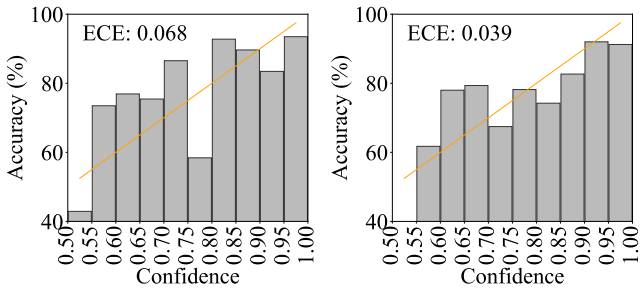
(a) Error (ϵ) $vs.$ Energy (σ) (b) Ours $vs.$ Traditional

Fig. 6: Proposed multi-objective optimization performance for the evaluated architectures. Measurements were made using our prototype (Fig. 5), with the *MAWIFlow* Jan. validation dataset, the results are reported with comparison to the baseline implementation without the utilization of early exits.

Genetic Algorithm (NSGA-II) [42] implemented on top of *pymoo* API. The NSGA-II uses a 100 population size, 100 generations, a crossover of 0.9, and a mutation probability of 0.1. These parameters were selected based on related works [43], and no significant differences were observed when they were varied. Throughout the experiments, the adequacy of the number of generations was verified by monitoring the objective improvements across iterations, ensuring that improvements plateaued before the execution of the final generation. The multi-objective feature selection aims to decrease energy consumption (σ) and error rate (ϵ). We compute the energy consumption through our prototype (Sec. VI), using our measurement approach described previously (Sec. VII-B). The selected DNNs are trained using the *MAWIFlow* January training dataset, while the objectives were measured through the January validation dataset. The resulting model's performance is measured through the testing dataset.

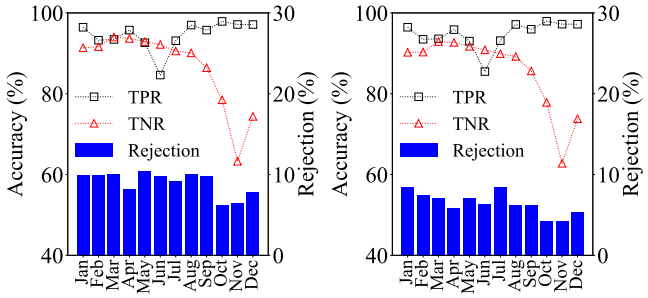
Figure 6a shows the Pareto curve of our proposed multi-objective optimization approach. The energy consumption is normalized according to the baseline with the traditional approach without using our proposed early exit strategy (Fig. 2). It is possible to observe a direct tradeoff between the proposal error rate and the resulting energy consumption at the *End Device*. In practice, increasing the number of events accepted at the 1st branch leads to a higher system error rate but also simultaneously reducing energy consumption. For example, our multi-objective optimization can achieve a $\approx$ 7% error rate while demanding only $\approx$ 11% of the energy consumption compared to the traditional approach. In addition, energy consumption can be reduced to as little as 1% if a 10% of error rate is tolerated. This shows that using NSGA-II to optimize our proposal's objectives effectively balances both goals simultaneously. Consequently, the network operator gains the flexibility to select an operation point that aligns with the system's specific requirements—prioritizing lower error rates if higher energy consumption is acceptable, or opting for reduced energy consumption at the cost of an increased rate of false alarms.

We further investigate how our proposed multi-objective optimization can enhance system performance. In this case,



(a) MobileNet Without Calibration (1st Branch) (b) MobileNet With Calibration (1st Branch)

Fig. 7: Reliability histograms for the 1st MobileNetV2 DNN branch on *MAWIFlow* January validation dataset.



(a) AlexNet (b) MobileNet

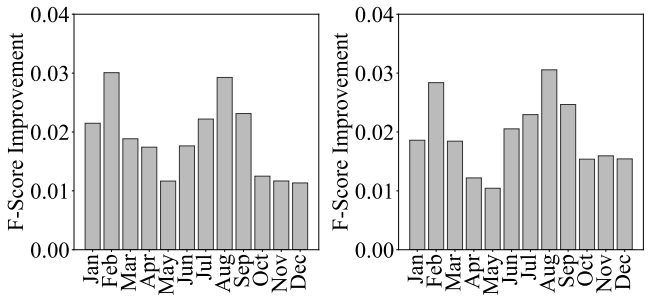
Fig. 8: Accuracy and rejection performance of our model as time passes on *MAWIFlow* dataset. DNNs are trained on *MAWIFlow* January training dataset. Inference uses our chosen operation point (Fig. 6a) and the proposed calibration scheme (Fig. 7).

as the operating point must align with the operator's requirements, we select an average operating point for the remaining evaluations (Fig. 6a, *Operation Point*).

Figure 6b compares our scheme's error and required processing resources with the chosen operation point $vs.$ the traditional approach without applying early exits. It is possible to observe a significant improvement in the required processing resources while still achieving a reduction in the error rate. In practice, our scheme can reduce average required processing resources (and thus, energy consumption) at the *End Device* in 86% and 44% while decreasing the error rate in 2.9% and 2.5% compared to traditional approaches for the AlexNet and MobileNetV2 DNNs, respectively.

As a result, the application of early exits can pave the way for energy-efficient EC, as our proposed model significantly reduces the energy consumption of resource-constrained devices. Moreover, this improvement in energy consumption is achieved without compromising the resulting model's error rate. We further investigate the trade-offs in classification delay and processing costs at the *Cloud Infrastructure* for the offloading task in Section VII-D.

Making use of the selected operation points (Fig. 6a), and before applying it through the entire *MAWIFlow* year, we conduct our proposed model calibration approach (see Sec. V-C, and Eq. 10). To achieve this objective, we apply a



(a) AlexNet (b) MobileNet

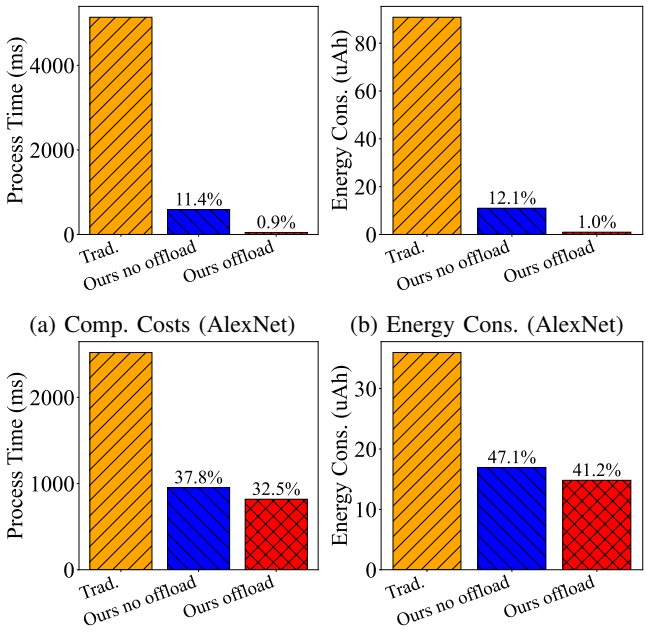
Fig. 9: F-Score performance improvement of our proposed scheme $vs.$ the traditional approach.

simple heuristic-driven search for each DNN branch to find the temperature T values for each confidence bin that improves the branch ECE value (Eq. 9). In practice, we consider a 10-sized bin for each branch (1st, and 2nd), and vary the temperature T from 0.0 to 1.0 in 0.00001 intervals. We then use the temperature T value for each bin that improves the resulting branch ECE.

Figure 7 shows the obtained reliability histogram for the 1st MobileNetV2 DNN branch on *MAWIFlow* January validation dataset, with $vs.$ without our proposed confidence calibration scheme. It is possible to observe that our proposed calibration model can approximate the classification confidence values for each bin to reflect the expected model accuracy. As an example, our proposal reduces the ECE on the 1st MobileNetV2 branch by 43% and when compared to its non-calibrated counterpart. Similarly, when both branches are considered, our calibration approach reduces the obtained ECE values by 38% and 40% for the AlexNet and MobileNetV2, respectively. Therefore, by adjusting the model's classification confidence values, we can reliably use them for early exits and our proposed rejection approach (see Section V-A).

To answer *RQ5*, we apply our proposed model throughout the entire *MAWIFlow* year. To achieve this objective, we use our chosen operation point obtained through our multi-objective optimization approach (Fig. 6a). In addition, the resulting model is passed through the confidence calibration procedure as shown in Figure 7. Similarly, the model is trained using the *MAWIFlow* January training dataset and evaluated as time passes without model updates.

Figure 8 shows our model's accuracy and rejection performance on the *MAWIFlow* dataset. Notably, the accuracy also degrades but not as significantly as observed with the traditional approaches (Fig. 8 $vs.$ 2). It is important to note that our proposal achieves the accuracy benefit in two ways. First, through the application of our proposed early exit technique, which offloads to the *Cloud Infrastructure* a subset of events that require additional processing capabilities. Second, through the classification with a reject option at the 2nd DNN branch, which identifies and suppresses potential misclassifications at the cloud, thereby preventing false alerts. This characteristic can be observed as time passes with the resulting model's rejection rate (Fig. 8, *Rejection Rate*). On average, throughout the entire *MAWIFlow* year, our proposal rejected 8% and 6% of



(a) Comp. Costs (AlexNet) (b) Energy Cons. (AlexNet)

(c) Comp. Costs (MobileNet) (d) Energy Cons. (MobileNet)

Fig. 10: Average inference computational time and energy consumption per event at the *End Device* with our proposed model. *Traditional* denotes the execution of the entire DNN at the *End Device* without using the early exit approach. *Ours (No offload)* denotes using the proposed approach with both exits processed at the *End Device*. *Ours (Offload)* denotes using the proposed approach, offloading the 2nd DNN branch to the *Cloud Infrastructure*.

events for AlexNet and MobileNetV2, respectively. Recalling that the operating point must be chosen according to the operator's needs. In this case, we could have further decreased the rejection rate, if required by the operator, by selecting an operating point with a higher error rate (Fig. 6a).

We further investigate the accuracy performance of our model when compared to the traditional approaches. Figure 9 shows the monthly F1-Score improvement of the selected DNN architectures with $vs.$ without our proposal. Our proposed scheme improves the F1-Score by an average of 0.02 for both selected architectures. As a result, our proposed scheme significantly reduces energy consumption for the inference task on the *End Device* while keeping or even improving classification accuracy. This improvement is achieved by applying early exits as implemented by our proposed model, which can lead to trade-offs due to the communication overhead with the cloud environment.

D. A Energy-efficient Edge Computing NIDS

To answer *RQ5* we analyze the performance of our proposal prototype (Fig. 5) in a distributed cloud environment. To achieve this objective, the *End Device* is deployed in the state of Paraná, in the southern region of Brazil, while the *Cloud Infrastructure* is deployed in two different IBM Cloud Services zones, namely *Brazil South* and *Central US*. The first deployment assesses the performance of our scheme with fewer network hops to the cloud, while the second evaluates

it with a higher number of network hops, which can lead to increased classification offloading latency.

Our first experiment investigates the computational processing benefits to the *End Device* achieved by offloading the inference task to the cloud. To achieve this objective, we implement our proposed prototype using the previously selected operation points (see Fig. 5) and offload the subset of events to the *Cloud Infrastructure* based on the specified acceptance thresholds. For this evaluation, we measure the average event processing time at the *End Device*, which includes both the execution of the 1st branch and the cloud offloading when required.

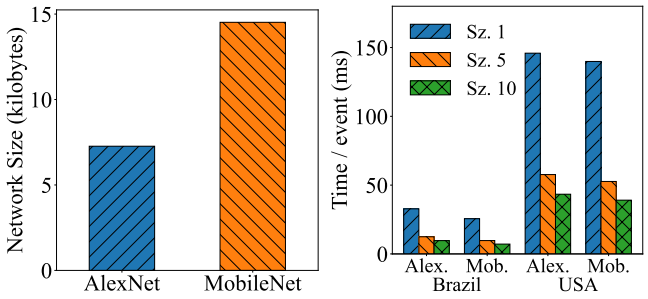
Figures 10a, and 10c show the average processing time per event with our proposed model *vs*. the at the device traditional approach. In this case, the *Traditional* denotes the execution of the device's traditional DNN architecture without early exits. It is possible to note that our proposed model substantially reduces the processing time at the *End Device* when compared to the traditional approaches. In practice, our scheme reduces processing costs to only 11% and 37% for the AlexNet and MobileNetV2, respectively, when both exits are executed at the *End Device* (*Ours (No Offload)*). In addition, if the 2nd DNN branch is offloaded to the cloud, the average event processing time is decreased to as little as 1% for AlexNet and to 32% for the MobileNetV2. This substantial difference between the selected DNN architectures is caused by the location where the 1st branch is introduced. While the 1st AlexNet branch is introduced after the first set of convolutional layers, the 1st MobileNet branch is introduced only after the 2nd bottleneck layer, thereby requiring additional processing before inference can terminate prematurely. Notwithstanding, given that approximately 90% of events are classified at the 1st DNN branch, which requires less processing, offloading a subset of events to the *Cloud Infrastructure* can reduce processing time compared to the traditional approach.

We further investigate the benefits achieved at the *End Device* with our proposed scheme by measuring the average energy consumption per event, using our measurement technique (see Section VII-B). Figures 10b, and 10d show the average energy consumption of our proposed model *vs*. the traditional on-device approach. Similarly, the energy consumption is substantially decreased compared to the execution of the entire inference process at the *End Device*. In this case, the energy consumption is decreased to only 12% and 47% if no offloading is used for the AlexNet and MobileNet, respectively. Conversely, if a cloud offloading strategy is used, energy consumption is reduced to 1% and 41% for AlexNet and MobileNet, respectively. Consequently, our proposed model significantly reduces processing costs and energy consumption at the *End Device*.

Our second experiment investigates the trade-offs when offloading events to the *Cloud Infrastructure*. To this end, we assess the average inference time per event according to the deployed *Cloud Infrastructure* zone. Table IV shows the average event inference time of our proposed scheme *vs*. the traditional on-device approach. In practice, our approach conducts event inference by an average of 28 and 223 milliseconds when deployed in *Brazil* (closer to the *End Device*), while

TABLE IV: Average event inference time of our proposed model according to the deployed *Cloud Infrastructure* zone.

DNN	Approach	Deploy Zone	Avg Event Inf. Time (ms)		
			Branch 1	Branch 2	Total
AlexNet	Ours	Brazil	18.08	10.28	28.36
		US	18.08	46.18	64.26
	Local	Device	20.24	1365.70	1385.94
MobileNet	Ours	Brazil	217.13	5.97	223.09
		US	217.13	32.6	249.73
	Local	Device	236.89	432.56	669.45



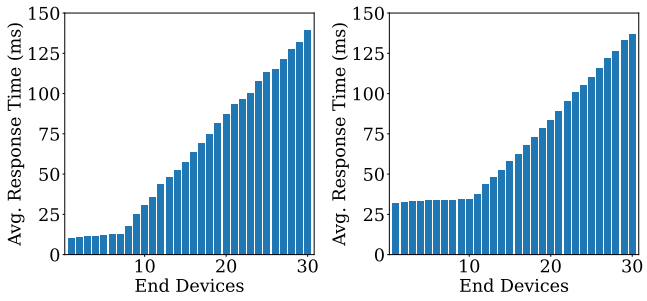
(a) Network usage per event (b) Proc. Time *vs*. Batch size

Fig. 11: Evaluation of the trade-offs incurred by offloading from *End Device* to *Cloud Infrastructure* the event classification (for each event reaching 2nd DNN branch). *End Device* is deployed in Brazil south region.

the traditional approach demands an average of 1385 and 669 milliseconds, a substantial reduction of 97% and 66% for the AlexNet and MobileNetV2.

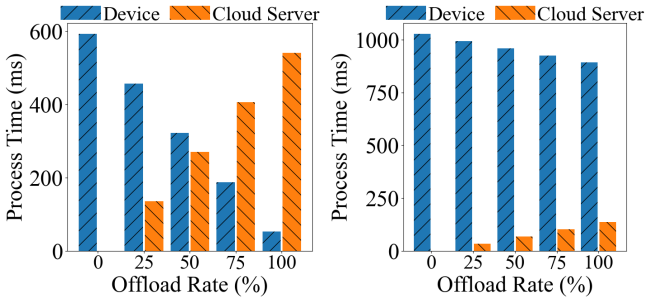
The deployment zone of the *Cloud Infrastructure* can impact the average event inference time when operating under real-time conditions. This observation holds true when comparing different deployment zones throughout the evaluation over the entire *MAWIFlow* dataset (Table IV, *Deploy Zone*). In this case, the deployment zone can increase the average inference time by 35 and 26 milliseconds for the AlexNet and MobileNetV2, respectively. This variation is primarily attributed to the differing RTT values of each deployment zone, showing that network conditions can significantly influence the resulting system processing time. During the experiments, the *Brazil* deployment zone (closer to the *End Device*) exhibited an average RTT of 23.51 ms, while the *US* deployment zone recorded an average RTT of 157.48 ms. This impact shows that the deployment zone, measured by *Brazil vs. US* deployment, incurs a difference of event inference time of 11% for MobileNet, reaching 56% for AlexNet. As a result, as fewer events are offloaded to the cloud, and the 1st DNN branch demands fewer processing costs, the overall resulting average event inference time is also decreased (Table IV). This characteristic suggests that the network operator should take into consideration the physical location of the *End Devices* when choosing to deploy the *Cloud Infrastructure*, as it can impact the execution time of the inference, and this is also architecture-dependent.

Our third experiment assess the network trade-offs caused by offloading the outputs of the 1st DNN branch to the *Cloud Infrastructure*. Recalling that for every event offloaded to the cloud our prototype demands the publishing of the 1st DNN



(a) Brazil (b) USA

Fig. 12: Evaluation of the scaling capabilities of our prototype with AlexNet on handling an increasing number of *End Devices*, from 1 to 30. Tests were executed using batches of size 10 while offloading all events to the Cloud Infrastructure.



(a) AlexNet (b) MobileNet

Fig. 13: Trade-off on process time *vs* offload rate to *Cloud Infrastructure*. The offload rate must be defined based on the operator's needs and can be adjusted accordingly.

branch output to a RabbitMQ topic (see Fig. 5). Fig 11a shows the network usage for each classification event offloaded to the cloud. In practice, our prototype requires $\approx$ 7.2 KB, and $\approx$ 14.5 KB for offloading each event from *End Device* to the *Cloud Infrastructure* for the AlexNet and MobileNetv2 DNNs respectively. It is important to note that this trade-off occurs only for that subset of events reaching the 2nd DNN branch. In this case, the chosen operation point (highlighted in Fig. 6a) offloads only $\approx$ 10% and $\approx$ 8% of events on the entire dataset for the AlexNet and MobileNet DNNs respectively. In addition, each network flow (event) comprises $\approx$ 11.6 network packets in *MAWIFlow* dataset (see Table II). Consequently, our proposal substantially decreases the network overhead compared to the traditional offloading strategy, wherein all network packets must be offloaded to the cloud.

We also evaluate how our proposal's average event processing time in the cloud can be further decreased when events are offloaded. To this end, we consider a scenario wherein events are offloaded following a batch-oriented rationale. In essence, we offload computation when a batch of N events are available, reducing the average network communication time for publishing the RabbitMQ messages and conducting batch-based inference through Pytorch API at the *Cloud Infrastructure*.

Figure 11b shows the impact on the average event processing time when a batch-oriented approach is used. In this case, our prototype significantly reduces the processing time by up to 70% and 72% for the AlexNet and MobileNetv2 DNNs, respectively. This improvement is achieved by increasing the batch size, which can also be used in a high network packet latency scenario by reducing the number of packets sent to the cloud and simultaneously enhancing processing time. As a result, despite the trade-offs in network usage and processing delays caused by offloading, our approach significantly reduces the average event energy consumption and processing time. In addition, the network operator may use this strategy to address network conditions that are not reliable, such as high latency or intermittent packet loss, by adjusting the batch size to optimize the trade-off between network usage frequency and processing time.

Finally, we also evaluated the scalability of our prototype in handling an increasing number of *End Devices*. Figure 12 illustrates the system's average event processing time as the number of *End Devices* increases with our proposal implementing the AlexNet DNN model. Notably, the deployment zone significantly impacts the average event processing time due to variations in RTT values. The results indicate that our proposed model can efficiently handle increasing client device loads, supporting up to 8 devices in the Brazil deployment zone and 10 devices in the USA deployment zone without impact on average event processing time. It is important to highlight that this evaluation assumes all events are forwarded to the *Cloud Infrastructure*, differing from the operation setting where only approximately 10% of events are transmitted (see Fig. 6a). Moreover, additional factors such as offload rate, traffic levels, and RTT to the *Cloud Infrastructure* can further impact the response time. Notwithstanding, our model prototype supports dynamic cloud infrastructure scaling, enabling it to adapt to fluctuating network conditions and load demands over time. For instance, the network operator can deploy additional VMs within the *Cloud Infrastructure* to accommodate a higher number of *End Devices* as needed. This scalability ensures that the proposed model is well-suited for deployment in real edge computing environments, typically involving a large and growing number of *End Devices*.

E. Discussion

To pave the way for an energy-efficient EC environment, participant entities must cooperate to reduce the overall processing requirements. Our proposal aimed at simultaneously improving the energy efficiency and processing costs of DNN-based NIDSs through an early-exit implementation rationale for edge computing. This was achieved by processing a subset of events at the *End Device*, thus relieving the *Cloud Infrastructure* when no additional processing capabilities are required. For the subset of events where classification cannot be reliably conducted, the *End Device* requests assistance from the *Cloud Infrastructure*, leading to a reduction in the processing footprint on both entities.

Thanks to our early exit strategy, this cooperation between the *End Device* and the *Cloud Infrastructure* can be adjusted based on the operator's needs. For example, the *End Device* can execute both branches locally when sufficient processing

capabilities and energy are available. Conversely, it can offload all computation to the *Cloud Infrastructure* when necessary, thereby relieving the *End Device* of processing demands. Figure 13 shows the process time trade-off according to the adopted offload rate to the *Cloud Infrastructure*. Our scheme enables the dynamic adjustment of offload rate according to the current *End Device* capabilities. In addition, these benefits are achieved with minimal network communication overheads (Fig. 11a). Consequently, we demonstrate that a strategy integrating the *End Device*, which locally conducts inference for a subset of events, with the *Cloud Environment*, which provides additional processing capabilities, can effectively reduce the processing footprint of the overall system.

VIII. Conclusion

In this paper, we presented a new DNN-based NIDS through early exits that operate following an energy-efficient EC architecture. Our proposed framework aims to simultaneously alleviate the processing demands of edge resource-constrained devices and cloud infrastructures by splitting the DNN inference task. Easier-to-be-classified events are processed at the edge with lower processing demands using an intermediate DNN branch, whereas the inference task can be offloaded to the cloud when additional processing capabilities are required. We compared our approach against conventional on-device strategies, showing the benefits of cooperating with edge resource-constrained devices and the cloud to simultaneously reduce processing demands and energy consumption. The proposed model substantially reduces the edge device energy consumption and processing time while maintaining intrusion detection accuracy. Notwithstanding, by processing a subset of events at the edge, cloud computing resources are alleviated, reducing the overall system processing footprint.

Finally, as future work, we plan to optimize the location of introduced DNN early exits to reduce network usage further. In addition, we aim to leverage rejected events to incrementally adjust deployed DNN branches, thereby addressing the non-stationary behavior of network traffic.

The dataset and source code used throughout the paper's experiments are publicly available at https://github.com/jasimioni/ids-ee-offload and the full dataset used in https://github.com/jasimioni/ids-ee-offload/tree/main/dataset/full.

Acknowledgment

This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants nº 304990/2021-3, 407879/2023-4, and 302937/2023-4.

References

[1] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E. B. Fernandez, and N. Yoshioka, "Landscape of architecture and design patterns for iot systems," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 091–10 101, 2020.

[2] Crowdstrike, "Global threat report," 2024. [Online]. Available: https://www.crowdstrike.com/global-threat-report/

[3] E. C. P. Neto, S. Dadkhah, S. Sadeghi, H. Molyneaux, and A. A. Ghorbani, "A review of machine learning (ml)-based iot security in healthcare: A dataset perspective," *Computer Communications*, vol. 213, pp. 61–77, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366423003936

[4] J. Verma, A. Bhandari, and G. Singh, "inids: Swot analysis and tows inferences of state-of-the-art nids solutions for the development of intelligent network intrusion detection system," *Computer Communications*, vol. 195, p. 227–247, Nov. 2022. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2022.08.022

[5] S. H. Mekala, Z. Baig, A. Anwar, and S. Zeadally, "Cybersecurity for industrial iot (iiot): Threats, countermeasures, challenges and future directions," *Computer Communications*, vol. 208, p. 294–320, Aug. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2023.06.020

[6] S. M. Kasongo, "A deep learning technique for intrusion detection system using a recurrent neural networks based framework," *Computer Communications*, vol. 199, p. 113–125, Feb. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2022.12.010

[7] Y. Gu, Y. Yang, Y. Yan, F. Shen, and M. Gao, "Learning-based intrusion detection for high-dimensional imbalanced traffic," *Computer Communications*, vol. 212, p. 366–376, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2023.10.018

[8] X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian, "Model complexity of deep learning: a survey," *Knowledge and Information Systems*, vol. 63, no. 10, p. 2585–2619, Aug. 2021. [Online]. Available: http://dx.doi.org/10.1007/s10115-021-01605-0

[9] E. Ntizikira, L. Wang, J. Chen, and K. Saleem, "Honey-block: Edge assisted ensemble learning model for intrusion detection and prevention using defense mechanism in iot," *Computer Communications*, vol. 214, p. 1–17, Jan. 2024. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2023.11.023

[10] L. Kong, J. Tan, J. Huang, G. Chen, S. Wang, X. Jin, P. Zeng, M. Khan, and S. K. Das, "Edge-computing-driven internet of things: A survey," *ACM Computing Surveys*, vol. 55, no. 8, p. 1–41, Dec. 2022. [Online]. Available: http://dx.doi.org/10.1145/3555308

[11] J. Liu, Y. Gao, and F. Hu, "A fast network intrusion detection system using adaptive synthetic oversampling and lightgbm," *Computers amp; Security*, vol. 106, p. 102289, Jul. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2021.102289

[12] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, and L. Guo, "Computation offloading in mobile edge computing networks: A survey," *Journal of Network and Computer Applications*, vol. 202, p. 103366, Jun. 2022. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2022.103366

[13] H. Liu, S. Zhang, P. Zhang, X. Zhou, X. Shao, G. Pu, and Y. Zhang, "Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, p. 6073–6084, Jun. 2021. [Online]. Available: http://dx.doi.org/10.1109/TVT.2021.3076780

[14] R. G. Pacheco, R. S. Couto, and O. Simeone, "On the impact of deep neural network calibration on adaptive edge offloading for image classification," *Journal of Network and Computer Applications*, vol. 217, p. 103679, Aug. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2023.103679

[15] A. Shahraki, M. Abbasi, A. Taherkordi, and A. D. Jurcut, "A comparative study on online machine learning techniques for network traffic streams analysis," *Computer Networks*, vol. 207, p. 108836, Apr. 2022. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2022.108836

[16] R. R. dos Santos, E. K. Viegas, A. O. Santin, and P. Tedeschi, "Federated learning for reliable model updates in network-based intrusion detection," *Computers amp; Security*, vol. 133, p. 103413, Oct. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2023.103413

[17] H. Rahmath P, V. Srivastava, K. Chaurasia, R. G. Pacheco, and R. S. Couto, "Early-exit deep neural network - a comprehensive survey," *ACM Computing Surveys*, vol. 57, no. 3, p. 1–37, Nov. 2024. [Online]. Available: http://dx.doi.org/10.1145/3698767

[18] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, Dec. 2016. [Online]. Available: http://dx.doi.org/10.1109/ICPR.2016.7900006

[19] L. She, W. Wang, J. Wang, Z. Lin, and Y. Zeng, "Progressive supervised pedestrian detection algorithm for green edge–cloud computing," *Computer Communications*, vol. 224, p. 16–28, Aug. 2024. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2024.05.022

[20] Z. Ning, X. Kong, F. Xia, W. Hou, and X. Wang, "Green and sustainable cloud of things: Enabling collaborative edge computing,"

IEEE Communications Magazine, vol. 57, no. 1, p. 72–78, Jan. 2019. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2018.1700895

[21] A. Maia, A. Boutouchent, Y. Kardjadja, M. Gherari, E. G. Soyak, M. Saqib, K. Boussekar, I. Cilbir, S. Habibi, S. O. Ali, W. Ajib, H. Elbiaze, O. Erçetin, Y. Ghamri-Doudane, and R. Glitho, "A survey on integrated computing, caching, and communication in the cloud-to-edge continuum," *Computer Communications*, vol. 219, p. 128–152, Apr. 2024.

[22] Y. Guo, "A review of machine learning-based zero-day attack detection: Challenges and future directions," *Computer Communications*, vol. 198, p. 175–185, Jan. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2022.11.001

[23] M. Ge, N. F. Syed, X. Fu, Z. Baig, and A. Robles-Kelly, "Towards a deep learning-driven intrusion detection approach for internet of things," *Computer Networks*, vol. 186, p. 107784, Feb. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2020.107784

[24] V. Ravi, R. Chaganti, and M. Alazab, "Deep learning feature fusion approach for an intrusion detection system in sdn-based iot networks," *IEEE Internet of Things Magazine*, vol. 5, no. 2, p. 24–29, Jun. 2022. [Online]. Available: http://dx.doi.org/10.1109/IOTM.003.2200001

[25] H. Nandanwar and R. Katarya, "Deep learning enabled intrusion detection system for industrial iot environment," *Expert Systems with Applications*, vol. 249, p. 123808, Sep. 2024. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2024.123808

[26] S. Khandelwal and S. Shreejith, "Exploring highly quantised neural networks for intrusion detection in automotive can," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, Sep. 2023.

[27] L. Zhang, X. Yan, and D. Ma, "A binarized neural network approach to accelerate in-vehicle network intrusion detection," *IEEE Access*, vol. 10, p. 123505–123520, 2022. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2022.3208091

[28] R. Zhao, G. Gui, Z. Xue, J. Yin, T. Ohtsuki, B. Adebisi, and H. Gacanin, "A novel intrusion detection method based on lightweight neural network for internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 12, p. 9960–9972, Jun. 2022. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2021.3119055

[29] O. R. Sanchez, M. Repetto, A. Carrega, R. Bolla, and J. F. Pajo, "Feature selection evaluation towards a lightweight deep learning ddos detector," in *ICC 2021 - IEEE International Conference on Communications*. IEEE, Jun. 2021. [Online]. Available: http://dx.doi.org/10.1109/ICC42927.2021.9500458

[30] O. Abdel Wahab, "Intrusion detection in the iot under data and concept drifts: Online deep learning approach," *IEEE Internet of Things Journal*, vol. 9, no. 20, p. 19706–19716, Oct. 2022. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2022.3167005

[31] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, ser. CCS '21. ACM, Nov. 2021. [Online]. Available: http://dx.doi.org/10.1145/3474369.3486864

[32] X. Zhao, G. Huang, J. Jiang, L. Gao, and M. Li, "Task offloading of cooperative intrusion detection system based on deep q network in mobile edge computing," *Expert Systems with Applications*, vol. 206, p. 117860, Nov. 2022.

[33] H. Liu, S. Zhang, P. Zhang, X. Zhou, X. Shao, G. Pu, and Y. Zhang, "Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 6073–6084, 2021.

[34] A. Mourad, H. Tout, O. A. Wahab, H. Otrok, and T. Dbouk, "Ad hoc vehicular fog enabling cooperative low-latency intrusion detection," *IEEE Internet of Things Journal*, vol. 8, no. 2, p. 829–843, Jan. 2021. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2020.3008488

[35] M. Ayyat, T. Nadeem, and B. Krawczyk, "Classynet: Class-aware early-exit neural networks for edge devices," *IEEE Internet of Things Journal*, vol. 11, no. 9, p. 15113–15127, May 2024. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2023.3344120

[36] A. Bakhtiarnia, Q. Zhang, and A. Iosifidis, "Improving the accuracy of early exits in multi-exit architectures via curriculum learning," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2021. [Online]. Available: http://dx.doi.org/10.1109/IJCNN52387.2021.9533875

[37] R. G. Pacheco, F. D. Oliveira, and R. S. Couto, "Early-exit deep neural networks for distorted images: providing an efficient edge offloading," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dec. 2021. [Online]. Available: http://dx.doi.org/10.1109/GLOBECOM46510.2021.9685469

[38] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, May 2021. [Online]. Available: http://dx.doi.org/10.1109/SPW53761.2021.00009

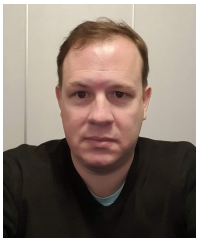
[39] MAWI, "MAWI Working Group Traffic Archive - Samplepoint F," 2021. [Online]. Available: https://mawi.wide.ad.jp/mawi/

[40] A. Moore, "Discriminators for use in flow-based classification," in *Dept. Comput. Sci., Univ. London, London, U.K., Rep. RR-05-13*, 2005.

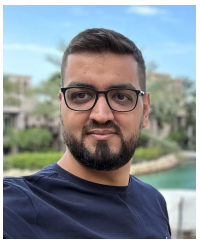
[41] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. of the 6th Int. Conf. on emerging Networking EXperiments and Technologies (CoNEXT)*, 2010.

[42] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[43] J. Zhang, B. Gong, M. Waqas, S. Tu, and S. Chen, "Many-objective optimization based intrusion detection for in-vehicle network security," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, p. 15051–15065, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1109/TITS.2023.3296002



Joao A. Simioni received the BS degree in Electrical Engineering from Federal University of Technology Paraná (UTFPR) in 2007 and is currently working toward the MS degree at PUCPR. His research interests include machine learning, edge computing and computer security.



Eduardo K. Viegas received the BS degree in computer science in 2013, the MSC degree in computer science in 2016 from PUCPR, and the PhD degree from PUCPR in 2018. He is currently an associate professor of the Graduate Program in Computer Science (PPGIa). His research interests include machine learning, network analytics and computer security.

Altair O. Santin received the BS degree in Computer Engineering from the PUCPR in 1992, the MSc degree from UTFPR in 1996, and the PhD degree from UFSC in 2004. He is a full professor of the Graduate Program in Computer Science (PPGIa) and head of the Security & Privacy Lab (SecPLab) at PUCPR. He is a member of the IEEE, ACM, and the Brazilian Computer Society.

Everton de Matos is a Lead Researcher at the Secure Systems Research Center (SSRC) of the Technology Innovation Institute (TII) in Abu Dhabi, United Arab Emirates. He holds a Ph.D. in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil, where he was also a Fulbright scholar at the University of Southern California (USC), USA. His research interests include information security, Internet of Things, embedded systems, and virtualization