Hardware-Aware Neural Architecture Search for Encrypted Traffic Classification on Resource-Constrained Devices

Adel Chehade, *Graduate Student Member, IEEE*, Edoardo Ragusa, *Member, IEEE*, Paolo Gastaldo, *Member, IEEE*, and Rodolfo Zunino

***Abstract*—This paper presents a hardware-efficient deep neural network (DNN), optimized through hardware-aware neural architecture search (HW-NAS); the DNN supports the classification of session-level encrypted traffic on resource-constrained Internet of Things (IoT) and edge devices. Thanks to HW-NAS, a 1D convolutional neural network (CNN) is tailored on the ISCX VPN-nonVPN dataset to meet strict memory and computational limits while achieving robust performance. The optimized model attains an accuracy of 96.60% with just 88.26K parameters, 10.08M floating-point operations (FLOPs), and a maximum tensor size of 20.12K. Compared to state-of-the-art (SOTA) models, it achieves reductions of up to 444-fold, 312-fold, and 15-fold in these metrics, respectively, significantly minimizing memory footprint and runtime requirements. The model also demonstrates versatility, achieving up to 99.86% across multiple VPN and traffic classification (TC) tasks; it further generalizes to external benchmarks with up to 99.98% accuracy on USTC-TFC and QUIC NetFlow. In addition, an in-depth approach to header-level preprocessing strategies confirms that the optimized model can provide notable performance across a wide range of configurations, even in scenarios with stricter privacy considerations. Likewise, a reduction in the length of sessions of up to 75% yields significant improvements in efficiency while maintaining high accuracy with only a negligible drop of 1-2%. However, the importance of careful preprocessing and session length selection in the classification of raw traffic data is still present, as improper settings or aggressive reductions can bring about a 7% reduction in overall accuracy. The quantized architecture was deployed on STM32 microcontrollers and evaluated across input sizes; results confirm that the efficiency gains from shorter sessions translate to practical, low-latency embedded inference. These findings demonstrate the method's practicality for encrypted traffic analysis in constrained IoT networks.**

***Index Terms*—Deep neural networks, encrypted traffic classification, hardware-aware neural architecture search, Internet of Things, resource-constrained devices.**

Received 15 January 2025; revised 22 August 2025 and 26 January 2026; accepted 15 February 2026. Date of publication 20 February 2026; date of current version 10 March 2026. This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. The associate editor coordinating the review of this article and approving it for publication was G. Sun. *(Corresponding author: Adel Chehade.)*

The authors are with the Department of Naval, Electrical, Electronic and Telecommunications Engineering (DITEN), University of Genoa, 16126 Genoa, Italy (e-mail: adel.chehade@edu.unige.it; edoardo.ragusa@unige.it; paolo.gastaldo@unige.it; rodolfo.zunino@unige.it).

Digital Object Identifier 10.1109/TNSM.2026.3666676

I. Introduction

THE proliferation of Internet of Things (IoT) technologies introduces security challenges that traditional methods often cannot handle effectively [1]. Resource-constrained devices generate huge amounts of data; relying on centralized servers to process that data may lead to transfer delays, increased network load, and additional power consumption [2]. Ideally, data flow monitoring should be carried out on edge devices to limit overhead in network management [3]. This paper proposes a design strategy to enable efficient traffic classification (TC) on constrained devices at the network edge.

The growth of encrypted Internet traffic, which now accounts for up to 96% of data [4], worsens this scenario. Widespread encryption complicates network security and traffic analysis [5], [6]; since modern protocols conceal packet contents, traditional methods such as Deep Packet Inspection (DPI) and port-based classification are mostly ineffective [3].

Efficient real-time traffic analysis is essential in today's cybersecurity ecosystem, especially for devices with limited resources like IoT and edge platforms [2]. These devices require fast, adaptive processing to manage encrypted traffic patterns and support real-time decision-making. Use cases include edge monitoring in high-bandwidth 5G applications, such as video streaming and gaming, which need smooth performance [7], and sensitive areas like finance and healthcare, where secure data exchange is critical [4].

Deep neural networks (DNNs) are increasingly used for traffic classification (TC) due to their high accuracy and ability to automatically extract features [5], [8]. However, their high computational needs limit their use in devices with limited resources [9]. This work explores ways to optimize DNN designs specifically for these restricted settings.

Beyond the DNN architecture, optimizing classification performance requires careful design choices. Network traffic can be classified at different levels, namely packets, flows, and sessions [10]. A packet is the smallest unit of data, while a flow includes packets sent from a source to a destination. A session extends this concept by including bidirectional communication between two endpoints, capturing the full packet exchange over time. This paper focuses on session-level classification, as it provides a broader view of network behavior, particularly for identifying encrypted traffic patterns [11], [12], [13], [14].

Another important aspect is the processing of packets; this process aims to ensure that only relevant information feeds the neural network. Several approaches in the literature [15],

[16], [17], [18] considered both accuracy and computational costs.

Finally, one should consider that the varying levels of complexity in the overall design problem can make certain strategies more feasible than others. This paper analyzes the interaction between those factors from an empirical viewpoint and provides the reader with practical design guidelines.

The paper presents a solution that relies on hardware-aware neural architecture search (HW-NAS) for network TC. HW-NAS has been applied in various deep learning (DL) fields [19], [20]; prior studies have shown that hardware objectives are most effectively handled during architecture optimization, rather than through post-hoc compression or manual tuning [21], [22]. In this work, HW-NAS is employed as a constraint-driven design methodology, in which parameter count, floating-point operations (FLOPs), and peak intermediate tensor size are explicitly bounded during the search. These constraints reflect practical deployment bottlenecks in model storage, computational cost, and runtime memory usage.

The application considered in this paper deals with a significant, uncovered aspect in the literature, that is, the design of optimized neural models that satisfy strict hardware constraints and yet maintain satisfactory classification performances across diverse network environments, thus making real-world deployment on IoT (edge) platforms feasible.

The main contributions of this paper are:

- **Hardware-aware deep networks for traffic classification:** To the best of our knowledge, this is the first work to design hardware-efficient deep networks for TC under strict, IoT-compatible constraints. HW-NAS specifically targets session-level classification and endows DNNs with hardware efficiency and satisfactory classification accuracy; previous approaches primarily focused on maximizing accuracy [5], [11], [13], [14], [23], [24], [25].
- **Header-level preprocessing and session length reduction:** This work extends existing studies by covering multiple header-level preprocessing methods, such as IP/MAC addresses, UDP padding, and anonymization, alongside session-length reductions, on the optimized model. Experimental evidence demonstrates that the model achieves stable performance, maintaining high accuracy with session lengths reduced by up to 75%, while achieving greater efficiency for resource-constrained devices through these reductions. At the same time, excessive data obfuscation or aggressive reductions in session lengths can affect accuracy; hence, careful selection of configurations is required to avoid drops of up to 7%. The paper gives practical guidelines for optimizing preprocessing and session length, providing a basis for future research on trading off accuracy and resource efficiency.
- **Efficiency benchmarks:** Established benchmarks validate the HW-NAS-driven strategy against state-of-the-art (SOTA) methods. The proposed model yields substantial resource savings, with reductions of up to 444-fold in parameters, 312-fold in FLOPs, and 15-fold in maximum tensor size compared to baseline models. These reductions streamline session-level TC by reducing computational cost and memory usage.
- **Comprehensive validation across tasks:** The HW-NAS-optimized model, tailored on the ISCX VPN-nonVPN dataset [26], demonstrates strong performance across diverse TC tasks. It achieves 96.60% accuracy in the main VPN-nonVPN classification. Additionally, it delivers high accuracy in VPN differentiation (99.86%), VPN-type (99.14%), broader traffic categories (96.74%), and session-level application identification (94.18%), showing its flexibility and effectiveness in real-world scenarios.
- **Cross-dataset generalization:** The very same architecture, tailored on the ISCX dataset, proves able to achieve up to 99.98% accuracy on the USTC-TFC2016 [27] and QUIC NetFlow [28] datasets.
- **On-device inference:** Efficiency gains from reduced session lengths also result in lower latency and energy consumption, enabling the model to run reliably on constrained IoT hardware; the quantized architecture operates in real time on Cortex-M4 and M7 microcontrollers, with latency under 115 ms and energy below 29 mJ.
- **Code availability:** The proposed HW-NAS is available at https://github.com/SEAlab-unige/ProtectIT_Unige.

II. Related Work

A. Traditional Approaches

Traditional TC methods such as port-based classification and DPI often fail on encrypted and obfuscated traffic.

Port-based classification associates traffic with specific services based on port numbers, which was suitable for older architectures but fails in modern networks due to dynamic ports and HTTP tunneling [3], [4]. Foundational studies indicated that default ports in Peer-to-Peer (P2P) protocols account for less than 70% of actual traffic, thus confirming the increasing ineffectiveness of this method [29], [30].

DPI identifies applications by detecting signatures within packet payloads [3], [4]. While effective on unencrypted traffic, it fails on encrypted flows where payloads are inaccessible [3], and it remains resource-intensive [4].

B. Machine Learning Methods

Classic machine learning (ML) approaches infer patterns from statistical features (e.g., packet size, flow duration, and arrival intervals) without examining packet payloads [3], [4].

Various algorithms, including Naïve Bayes for real-time application classification, k-Nearest Neighbors (k-NN), and Decision Trees, have demonstrated effectiveness in identifying encrypted flows based on these features [26]. Additionally, Random Forests (RF) have been successfully employed for TC in real-world specialized environments like software-defined networking (SDN) [31].

Addressing the need for high-speed processing, [32] implemented a fully in-switch classification framework that executes RF inference at line rate within the data plane. Moreover, [33] combined flow label propagation with compound classification to detect unknown flows and boost accuracy.

Some limitations affect traditional ML methods for TC. They rely on manual feature extraction, which is labor-intensive, and even strong models such as RF require frequent retraining to maintain satisfactory performance [4].

DL models extract complex features from raw traffic, bypassing manual feature selection. While Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs)

achieve high accuracy, they are often computationally heavy, hindering deployment in resource-limited applications.

Several studies focused on session-level data. In [11], a 1D-CNN and 2D-CNN were evaluated on this task: the 1D-CNN scored an accuracy of 86.6% and outperformed the 2D-CNN. The approach presented in [27] also used 1D-CNN for malware TC. [34] proposed an image-based approach that converted initial non-zero payload sessions into grayscale images for classification using CNNs. Although it achieved high F1 scores (97.73% for conventional encrypted traffic and 99.55% for VPN traffic), the coarse-grained handling of subflows disregarded relevant session information. The framework in [35] combined CNN and Stacked AutoEncoders (SAE) for encrypted traffic classification (ETC) by integrating trimmed raw traffic data and statistical features; the method reached 98% F1-score, at the expense of additional time required for feature extraction. Long Short-Term Memory (LSTM) networks, often used to model sequential dependencies, were employed in the Inception-LSTM (ICLSTM) model [24], which converted traffic data into grayscale images and combined Inception modules with LSTM layers; it attained over 98% accuracy but showed limitations in adapting to diverse real-world scenarios. The integration of LSTM, CNN, and a squeeze-and-excitation module to capture spatiotemporal features [13] achieved over 90% accuracy on encrypted and unencrypted traffic via end-to-end representation learning. Focusing on attention mechanisms added computational overhead and limited scalability for resource-constrained environments.

At the packet level, [6] introduced *Deep Packet*, which classified encrypted traffic using a combination of SAE and CNN. Packet-level approaches require large volumes of data, thus leading to high computational requirements and extended training times. The work in [36] proposed *DataNets*, which used Multilayer Perceptron (MLP), SAE, and CNN models for the accurate categorization of encrypted traffic in smart home networks. Aggressive undersampling was used to address dataset imbalance, but it may affect the model's generalization performance. The authors in [37] proposed a deep hierarchical neural network for packet-level malicious traffic detection; a 1D-CNN for spatial feature extraction and Gated Recurrent Unit (GRU) for temporal features scored high accuracy levels across multiple intrusion detection datasets [38], [39].

Several works integrated session and packet-level data. Capsule Networks (CapsNet) [23] yielded a session packet-based model for ETC. It prioritized classification performance at the expense of computational overhead. Likewise, the CBS model in [5] combined 1D-CNN, attention-based Bi-LSTM, and SAE for ETC at both session and packet levels by integrating spatial, temporal, and statistical features. CBS employed Generative Adversarial Networks (GANs) for data augmentation and class imbalance handling, resulting in improved performance. The computational requirements limited its use in offline processing and made the method unsuitable for real-time applications. In [40], the authors proposed a hybrid model that extracts flow-level statistical features and classifies them using a deep feedforward DNN followed by a maximum entropy classifier. Despite reporting 99.23% accuracy, the approach relies on handcrafted features and static encoding, limiting its adaptability to raw or evolving traffic. Other works [18] applied attention-based LSTM and Hierarchical Attention Networks (HAN) to classify flows as time-series data. The attention mechanism captures key features, achieving 91.2% accuracy, but with limited efficiency on constrained devices.

C. Trends in Hardware-Aware NAS

NAS methods improve model performance by tuning the network architecture on the target data. Many approaches have been proposed in the literature [19], [20], each improving architecture exploration in unique ways.

HW-NAS extends NAS by optimizing models for accuracy alongside hardware constraints, such as memory, latency, and computational efficiency [41]; this direction was established by works such as MnasNet [21] and ProxylessNAS [22], which integrated hardware objectives into the search process targeting mobile platforms. Frameworks like MCUNet [42] further specialized these techniques for microcontrollers by using system-algorithm co-design to fit deep models within strict SRAM and Flash constraints. Additionally, studies such as [43], [44], and [45] illustrated how HW-NAS could produce models tailored to efficient deployment on resource-limited platforms by matching network design with target hardware needs.

A recent work applied NAS to ETC [8], although without strict IoT-centric constraints. Other works have applied NAS to malware detection using proximal-iteration search [46], and have shown that session-level ETC is feasible under hardware constraints [47]. A HW-NAS strategy was also adopted in [48] for session-level inference on microcontrollers; however, the study did not account for how input data variation can affect efficiency and performance.

NAS effectiveness relies on careful adaptation to the specific constraints and priorities of the target application domain. The research presented in this paper relies on HW-NAS to develop a hardware-efficient neural network for ETC. The target environment includes IoT applications, in which high accuracy and resource efficiency are of paramount importance.

III. Preliminaries

NAS automates neural network design by applying a search algorithm to identify optimal architectures within a predefined search space; this approach outperforms traditional handmade models [8], [20] in terms of accuracy.

Conventional NAS algorithms aim to optimize computational performance but often overlook hardware constraints; as a result, the models they produce may lack the efficiency required for their deployment in resource-limited devices. HW-NAS overcomes this limitation by directly including hardware constraints in the search process; this makes it possible to envision deployment on a variety of application devices, such as microcontroller units (MCUs), field-programmable gate arrays (FPGAs), and cloud accelerators. The extended optimization problem takes into account multiple objectives and constraints, including accuracy, memory usage, computational complexity (measured in FLOPs), and energy efficiency [19].

The HW-NAS optimization problem is to identify an architecture $a \in \mathcal{A}$ that minimizes the validation loss on a given dataset $\mathcal{D}$. This objective can be formally defined as:

$$\begin{aligned} \min_{a \in \mathcal{A}} \quad & \mathcal{L}_{\text{val}}(w^*(a), a) \\ \text{s.t} \quad & w^*(a) = \operatorname*{argmin}_{w} \mathcal{L}_{\text{train}}(w, a) \\ & \psi(a, HW) < \text{Thr} \end{aligned} \tag{1}$$

where $\mathcal{A}$ denotes the search space containing all candidate architectures, $\mathcal{L}_{\text{val}}$ represents the validation loss, and $\mathcal{L}_{\text{train}}$ is the training loss minimized to optimize the model weights $w^*(a)$. $\psi(a, HW)$ is a function that measures the performance of the architecture with respect to hardware-specific metrics and Thr is a threshold value associated with these constraints.

Three main phases characterize NAS, namely, defining the search space, selecting the search strategy, and evaluating the performance. The search space defines the set of admissible network structures (including layer configurations and optimization hyperparameters), which collectively determine the exploration boundaries for the NAS algorithm. The search strategy then controls how the NAS algorithm explores the architecture space. Finally, the performance evaluation tests the candidate architecture using a task-specific merit function, which adapts to the requirements of the application.

IV. Methodology: Design of Efficient Network for Traffic Classification

The method described in this paper provides a novel design pipeline to develop lightweight, optimized DNN architectures for session-level ETC, with a focus on deployment on resource-constrained IoT devices. The workflow consists of three key stages: (1) **Data preprocessing** arranges raw traffic data for DL by converting sessions into a consistent input format; (2) in **HW-NAS execution**, the search algorithm explores the space of admissible architectures to find a high-performing model that also satisfies hardware constraints; finally, (3) the **Model selection and testing** first picks out the best-performing architecture, based on performance and constraints, then completes a rigorous testing process involving multiple tasks and deployment settings, ensuring generalization and real-world feasibility.

A. Preprocessing

The following steps implement the preprocessing phase.

Session extraction: Raw traffic data is divided into sessions; each session consists of an ordered sequence of bidirectional packets exchanged between two endpoints, identified by IP addresses, port numbers, and protocol. Packet data is typically stored as raw bytes.

Data cleaning: This step removes data-link layer information, including MAC addresses, and anonymizes IPs. This approach is frequently adopted in the literature to prevent overfitting; otherwise, some session-specific features could lead the model to memorize traffic patterns instead of learning generalization-relevant features [5], [6].

Filtering irrelevant data: Packets without payloads (e.g., those featuring SYN, ACK, or FIN flags) and irrelevant DNS segments are discarded. Prior studies showed that those kinds of packets did not contribute meaningfully to TC and could actually add noise to the dataset. Filtering these packets ensures only meaningful traffic is used for training [5].

Session normalization: In compliance with a method originally proposed in [11] and [27], each session is normalized to a uniform length of 784 bytes. This approach proved effective in handling varying session sizes while ensuring consistency for model input. Longer sessions are truncated, and shorter sessions are padded with null bytes to maintain a uniform length over all sessions.

Data scaling: This process prepares the data for subsequent input to the model, converting raw byte information into a standardized format suitable for neural network training. Each byte in the raw packet values forming session data is normalized into the range [0,1].

B. Optimization Problem in HW-NAS

The HW-NAS design strategy adopted in this paper casts architecture selection as a constrained optimization problem over candidate models $a \in \mathcal{A}$, defined as follows:

$$\begin{aligned} \min_{a\in\mathcal{A}} \quad & \text{Accuracy}_{\text{val}}(w^*(a), a) \\ \text{s.t} \quad & w^*(a) = \underset{w}{\text{argmin}}\ \mathcal{L}_{\text{train}}(w, a) \\ & |P(a)| < F_{\text{Th}} \\ & |T(a)| < R_{\text{Th}} \\ & \text{Flops}(a) < \text{Flops}_{\text{Th}} \end{aligned} \tag{2}$$

Hardware constraints are applied using thresholds F_{Th}, R_{Th}, and Flops_{Th}, which manage the critical resource limitations typically characterizing IoT devices. These thresholds stem from existing works, such as [43].

The parameter-count threshold, F_{Th}, controls the total number of model parameters $P(a)$ of a candidate architecture and directly affects memory usage. Reducing that quantity minimizes the model's memory footprint and helps fit the limited Flash memory available on edge devices. The threshold on the maximum tensor size, R_{Th}, bounds the peak intermediate activation size $T(a)$ during inference and ensures that it fits the available RAM. This is critical for session-level classification tasks, where efficient memory utilization is essential to avoid overflows during computation. Finally, the FLOPs threshold, Flops_{Th}, limits computational complexity by bounding the number of floating-point operations, which serves as a proxy for inference speed and energy consumption. This constraint is essential for maintaining real-time performance on devices that often lack dedicated Graphics Processing Units (GPUs).

C. Search Space Design

The HW-NAS search space in this work is a block-wise structure relying on 1D CNNs, which efficiently capture dependencies in session-based traffic data. They extract localized structural patterns within packet content (e.g., payload and headers) and sequential trends across an entire session (e.g., bidirectional exchanges). Recurrent neural networks (RNNs), such as LSTMs and GRUs, can model sequential dependencies in session-based traffic data. At the same time, their step-by-step processing paradigm increases overhead and training time, making them unsuitable for real-time applications. Likewise, Transformers capture global relationships using self-attention but exhibit impractical memory and computational requirements for resource-constrained environments. In contrast, 1D CNNs can benefit from weight sharing and local connectivity to extract features efficiently, yielding a balance between computational cost and classification accuracy.

Each convolutional layer in the CNN block is defined by its kernel size, number of filters, padding, and stride. These parameters allow the model to flexibly adjust its receptive field and depth so that it can capture both detailed packet-level

features and overall session-level patterns. A series of batch normalization, ReLU activation, pooling (max or average, with adjustable pooling size), and dropout (with a configurable dropout rate) follow each convolutional layer. This improves feature representation while minimizing computational load. A Global Average Pooling (GAP) layer supports the final feature aggregation, which condenses spatial information into a compact representation and feeds a dense layer for the eventual classification outcome.

D. Evolutionary Algorithm for NAS Search

In the proposed HW-NAS framework, an evolutionary algorithm supports the exploration of architecture space. Evolutionary algorithms proved to be highly effective toward that purpose [19]. This approach is suitable to ensure that target architectures meet predefined hardware requirements [43].

The search strategy starts from an initial (parent) architecture, a_0. A mutation function $R_m(a_p)$ then spawns candidate architectures (children). Mutations involve random modifications to the parent, such as adding new blocks with randomly generated hyperparameters, removing blocks to reduce complexity, or adjusting parameters within existing blocks (e.g., number of filters, kernel size). Each child is then evaluated against the predefined hardware constraints $\mathcal{H}$, and only those that satisfy them proceed to training.

This spawning/selection process iterates over a predefined number of generations N_g; each generation yields N_c child architectures. If a child fails the hardware constraint check, further mutations are applied until the required number of admissible children is achieved. A maximum depth constraint limits the number of blocks within each architecture, ensuring that each child remains within a preset complexity level.

Algorithm 1 Evolutionary NAS with hardware constraints.

Inputs: Training set $\mathcal{X}_\mathcal{T} = \{X_i, y_i\}_{i=1...n}$, validation set $\mathcal{X}_\mathcal{V} = \{X_i, y_i\}_{i=1...m}$, search space $\mathcal{A}$, initial parent a_0, mutation operator $R_m(\cdot)$, N_c children per generation, number of generations N_g, evaluation function $E(a, \mathcal{X}_\mathcal{V})$, hardware constraints $\mathcal{H}$.

Procedure:

Initialize parent architecture $a_p \leftarrow a_0$
Initialize best architecture $a^* \leftarrow a_p$
  for $g = 1$ to N_g **do**
    for $c = 1$ to N_c **do**
    Generate child $a_c \leftarrow R_m(a_p)$
    if a_c satisfies $\mathcal{H}$ **then**
      Train a_c on $\mathcal{X}_\mathcal{T}$
    end if
  end for
  Select $a_p \leftarrow \arg\max_{a_c} E(a_c, \mathcal{X}_\mathcal{V})$
Update $a^* \leftarrow \arg\max(E(a^*, \mathcal{X}_\mathcal{V}), E(a_p, \mathcal{X}_\mathcal{V}))$
end for
Return a^*

The search loop is outlined in Algorithm 1. After passing the hardware check, each child undergoes training on the training dataset $\mathcal{X}_\mathcal{T}$, and is subsequently evaluated for validation accuracy on the validation set $\mathcal{X}_\mathcal{V}$. The architecture with the highest validation accuracy becomes the new parent, a_p, for the next generation. After all generations, the architecture with the highest validation score is selected.

TABLE I
ISCX VPN-NONVPN 2016 NUMBER OF SAMPLES

Category	Application	Encapsulation		Category Ratio (%)
		Non-VPN	VPN	
Chat	aim chat, facebook, hangout, icq, skype	7121	4014	25.98
Email	email, gmail	5183	293	12.78
Streaming	netflix, vimeo, youtube, spotify	1464	474	4.52
File Transfer	skype, scp, sftp, ftp	1251	986	5.22
VoIP	skype, hangout, facebook, voipbuster	5409	16184	50.39
P2P	bittorrent	-	475	1.11

TABLE II
OVERVIEW OF CLASSIFICATION TASKS

Experiment	Description	Classes
VPN-NonVPN	VPN and Non-VPN traffic classification	11
VPN-Diff	Protocol encapsulation detection	2
VPN-Type	VPN traffic type classification	6
NonVPN-Type	Non-VPN traffic type classification	5
Traffic-Cat	Network usage categorization	6
App-ID	Application identification	15

V. EXPERIMENTAL SETUP

This section outlines the empirical setting for designing and evaluating efficient DNNs for session-level TC. It covers: (1) the encrypted traffic datasets; (2) the deployment-driven hardware constraints; and (3) the HW-NAS process for discovering models under resource limits.

A. Datasets and Tasks

1) Encrypted VPN Dataset: The ISCX VPN-nonVPN dataset [26], consisting of approximately 30GB of traffic divided into 11 classes, is used in this study. It includes traffic for various applications in packet capture (PCAP) format, labeled according to application and activity type. Table I outlines the dataset structure, detailing traffic categories, encapsulation types (VPN and Non-VPN), and their respective proportions. Widely adopted in TC research [5], [6], [11], [14], [18], [24], this dataset serves as the main benchmark for architecture optimization and supports the definition of various TC tasks.

These tasks span a range of complexities and objectives, as detailed in Table II. The rows represent different tasks, while the columns specify the name of the experiment, its objective, and the number of output classes. VPN-NonVPN classifies traffic into 11 categories across both VPN and Non-VPN traffic. VPN-Diff performs binary classification to detect whether traffic is VPN or Non-VPN. VPN-Type and NonVPN-Type further classify traffic types within the VPN and Non-VPN groups into 6 and 5 categories, respectively. Traffic-Cat evaluates general traffic categorization across both VPN and Non-VPN traffic into 6 usage-based categories. Finally, App-ID identifies 15 specific network applications.

TABLE III
USTC-TFC2016 AND QUIC NETFLOW OVERVIEW

Dataset	Category	Applications	Total Samples (% of dataset)	# Classes
USTC-TFC2016	Benign	BitTorrent, Facetime, FTP, Gmail, MySQL, Outlook, Skype, SMB, Weibo, WoW	405,517 • 81.0%	20
	Malware	Cridex, Geodo, Htbot, Miuref, Neris, Nsis-ay, Shifu, Tinba, Virut, Zeus	95,334 • 19.0%	
QUIC NetFlow	Chat	Google Hangouts Chat	14,528 • 8.6%	5
	Voice Call	Google Hangouts VoIP	61,311 • 36.3%	
	File Transfer	File Transfer Service	5,045 • 3.0%	
	Video Streaming	YouTube	49,056 • 29.1%	
	Music Streaming	Google Play Music	36,466 • 21.6%	

TABLE IV
TARGET MICROCONTROLLER SPECIFICATIONS

Specification	STM32F746G-DISCO	Nucleo-F401RE
Core	Arm Cortex-M7	Arm Cortex-M4
Flash Memory	1 MB	512 KB
RAM	340 KB	96 KB
Clock Speed	216 MHz	84 MHz
Power Supply	5V	5V

2) Generalization Benchmarks: In addition to ISCX, we consider two public benchmarks, USTC-TFC2016 [27] and QUIC NetFlow [28], to evaluate generalization under different traffic distributions. Both datasets are provided in raw pcap format (approximately 4GB for USTC and 120GB for QUIC) and reflect distinct classification scenarios.

USTC-TFC2016 includes 20 traffic classes across benign and malware categories, spanning diverse protocols and communication behaviors in both encrypted and unencrypted data.

QUIC NetFlow focuses on encrypted services over the QUIC protocol, grouped into five usage categories: chat, voice, video, music, and file transfer. It captures structural differences in encrypted traffic generated by the QUIC protocol, which encrypts transport-layer metadata over UDP.

Table III summarizes the key characteristics of both datasets. Each row specifies a traffic category, the corresponding application type, the number and share of samples, and the total number of classes per dataset.

B. Target Hardware for Deployment

We target two microcontroller platforms commonly used in embedded systems: the STM32F746G-DISCO and the Nucleo-F401RE. These boards represent distinct deployment scenarios for low-power TC at the edge: the STM32F7 suits moderately capable systems, while the Nucleo-F4 is a good example of a minimal IoT edge node.

Table IV reports key specifications for both targets. Rows list core type, Flash/RAM, clock speed, and power. These constraints set a reference for the deployment envelope and guide the design of lightweight models for on-device inference.

C. HW-NAS Procedure

The HW-NAS search is conducted on the VPN-NonVPN task from ISCX, selected for its comprehensive nature and frequent use in prior research, which provides a solid foundation for comparisons with SOTA models. It includes both VPN and Non-VPN traffic, capturing shared structural patterns relevant to broader classification settings. The goal is to discover a network architecture that generalizes effectively across various tasks and datasets, without the need for task-specific designs.

We ran the search on a workstation with an NVIDIA 2080 Ti GPU, using session-level data preprocessed with Scapy, while the HW-NAS framework was built on Keras and TensorFlow.

The thresholds for memory usage and FLOPs have been initially set using the minimum values observed among session-level TC models in prior works as reference. These metrics were calculated following details provided by each study and implemented in Keras to ensure consistency and accuracy. We found substantial headroom for improving efficiency. Since deployment targets both STM32F746G-DISCO and the more constrained Nucleo-F401RE, the thresholds were aligned with the latter's resource limits (see Table IV). Specifically, we set the maximum parameter count to 120K and the maximum tensor size to 22K, corresponding to 480 KB Flash and 88 KB RAM under 32-bit floating-point assumptions (float32). These estimates match the Nucleo's 512 KB Flash and 96 KB RAM and enable compatibility checks during float32-based training and search. The FLOPs threshold was set to 11M based on the device's 84 MHz clock, which theoretically allows up to 84 million operations per second. This implies an upper bound of approximately 130 ms for inference execution, excluding memory access overheads, which are not explicitly modeled in this estimate. This latency remains acceptable in session-level TC scenarios, where inference is invoked once per aggregated session, and it falls well within typical limits observed in edge-hosted analytics and network monitoring tasks, which tolerate response times between 100-250 ms depending on application domain [49]. Further compression (e.g., quantization or pruning) can provide additional margin at deployment. Empirical measurements on the target MCUs confirmed that latency remains within this limit, as reported in Section VIII.

A validation set consisting of 20% of the training data is created using a standard holdout method. Architectures are trained for up to 100 epochs, with an initial learning rate of 10^{-3}, a batch size of 128, and a learning rate reduction triggered by a plateau in validation loss. Early stopping is applied using the validation loss as the stopping criterion. Networks are trained 3 times using a multi-start approach, and the best validation accuracy is used to score each architecture.

HW-NAS is conducted over 100 generations with 10 candidate architectures (children) per generation. On average, 16 children had to be generated per generation to obtain 10 admissible ones, as approximately 6 were discarded due to hardware constraint violations. Each child is derived through two random mutations applied to the parent, with mutation types drawn uniformly from block insertion, block removal, or block-level parameter modification (e.g., kernel size, filters). To prevent overcomplex architectures, the maximum depth is 5 blocks. This setup promotes structural diversity and reduces the risk of early convergence to suboptimal solutions.

The search space includes filters ranging from 16 to 140, kernel sizes from 3 to 7, strides from 1 to 6, and dropout rates from 0.1 to 0.5. Pooling, if enabled, can be either max or average, with a pool size between 2 and 3. Padding is

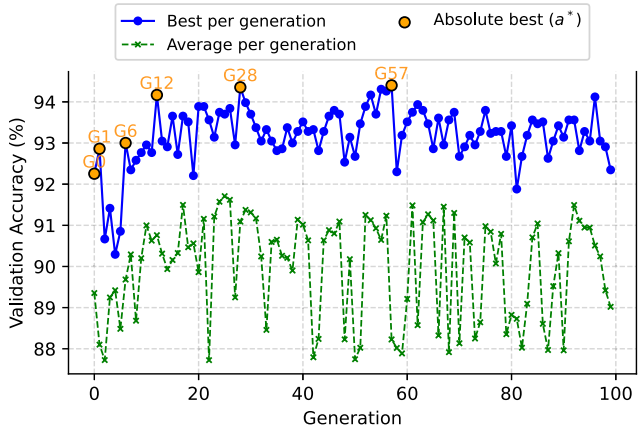

Fig. 1. Validation accuracy over HW-NAS generations.

TABLE V
BEST CNN ARCHITECTURE IDENTIFIED BY HW-NAS

#	Layer	Input Dim	Filt/Units	Ker/Pool	Str	Pad
1	Conv1D + ReLU	784 x 1	129	7	5	valid
2	Conv1D + ReLU	156 x 129	110	4	2	valid
3	AvgPool1D	77 x 110	-	3	2	same
4	Conv1D + ReLU	39 x 110	38	7	2	valid
5	MaxPool1D	17 x 38	-	2	2	same
6	GAP	9 x 38	-	-	-	-
7	Dense + Softmax	38	11	-	-	-

set as either "same" or "valid." The best-performing model is selected based on validation accuracy.

Figure 1 shows the validation accuracy of the best candidate per generation (solid line), along with the average accuracy of all admissible children (dashed line). Orange markers indicate generations where a new absolute best model was found (e.g., G0, G1, G6, G12, G28, G57); no further improvements occurred beyond G57, indicating that the search converged early under the imposed hardware constraints. The average curve complements this view by capturing the overall performance trend of each generation beyond the individual best models.

The best model found by HW-NAS is retrained on all tasks in Table II and evaluated on the external datasets in Table III to assess its generalization performance. For these retraining experiments, we performed 10 independent runs and reported the sample standard deviation of accuracy to capture performance variability. Each experiment is trained with early stopping, using a maximum of 50 to 200 epochs depending on the task and dataset, with higher limits applied when convergence required more iterations.

VI. RESULTS AND EVALUATION

A. HW-NAS-Optimized Architecture

The best architecture identified by HW-NAS is shown in Table V. Rows list the sequential CNN layers, and columns report the corresponding layer attributes: layer type (e.g., one-dimensional convolution (Conv1D), average pooling (AvgPool1D), max pooling (MaxPool1D), or Dense), input dimensions (Input Dim), number of filters or units (Filt/Units),

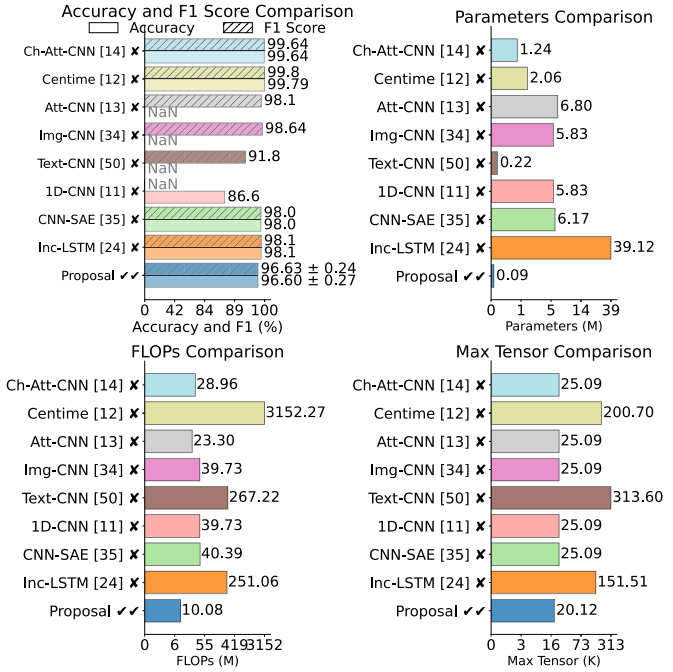

Fig. 2. Comparison of session-level methods based on accuracy, F1 score, parameters, max tensor, and FLOPs.

kernel or pooling size (Ker/Pool), stride (Str), and padding (Pad).

A key aspect is the progressive reduction of filters across convolutional layers, which allows the model to balance feature extraction and computational cost. Kernel size and stride vary adaptively: early layers use larger receptive fields to capture broad patterns, while deeper layers focus on finer details. This design achieves both accuracy and hardware efficiency, with only 88.26K parameters, a maximum tensor size of 20.12K, and 10.08M FLOPs. These values are comfortably below the thresholds set by the Nucleo-F401RE and STM32F7 platforms, confirming on-device deployability; they also indicate likely compatibility with other microcontrollers and low-power edge devices available on the market with similar or higher resource capacity.

In the following comparisons, we assess if SOTA models meet RAM and Flash limits using float32 parameter counts and max tensor sizes as proxies. Models above these limits may run on more capable platforms, but require structural reduction to operate on the tested low-power MCU class (e.g., redesign or compression). Furthermore, when a model fits within the memory constraints but exceeds the proposed model's 10.08M FLOPs, one should expect both latency and energy consumption to increase on comparable devices.

B. VPN-NonVPN Task: Performance and Efficiency

This subsection reports the session-level results of the VPN-NonVPN experiment, the target scenario of this study.

Figure 2 compares accuracy, F1 score, parameters, FLOPs, and maximum tensor size across SOTA models. The proposed model is annotated with its standard deviation across ten independent training runs. A single check mark (✓) indicates compatibility with STM32F7-class MCUs; a double check mark (✓✓) denotes compatibility with both STM32F7 and

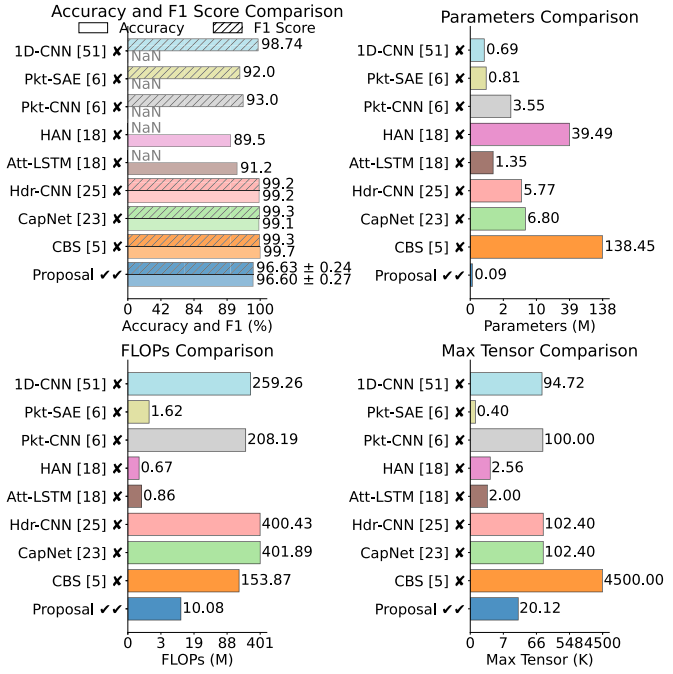

Fig. 3. Comparison of non-session methods based on accuracy, parameters, max tensor, and FLOPs.

the more constrained F401RE platform; a cross (×) marks cases where deployment is not feasible. No other model in this session-level comparison meets both constraints, underscoring the distinctive efficiency of the proposed architecture.

The proposed model achieves an accuracy of 96.60% and an F1 score of 96.63%, which are slightly lower than some of the most resource-intensive models in the SOTA. For example, [12] and [14] achieve accuracy and F1 scores exceeding 99% but require 2.06M and 1.24M parameters, respectively, compared to the proposed model's 0.09M, with a reduction of up to 22 times. Their FLOPs requirements (3152.27M and 28.96M) surpass the proposed model's (10.08M) by up to 312 times and nearly 2.9 times, respectively. The maximum tensor size in certain works, such as 313.60K in [50], further underscores the efficiency of the proposed model, which operates with a tensor size of only 20.12K, achieving a reduction of over 15 times. [24] achieves an accuracy of 98.10%, requiring an increment of 435, 24.9, and 7.5 times of the three hardware estimators. Similarly, [35] achieves 98.00% accuracy at the expense of 68, 4, and 1.25 times increment. These results highlight the proposed model's ability to deliver competitive performance compared to SOTA while remaining within the strict memory and compute constraints of microcontroller-class devices.

Figure 3 extends the analysis to non-session methods, which include packet-level [6], [51], flow-based [18], [25], and hybrid [5], [23] input approaches. These differ in format, preprocessing, and problem framing, making direct comparisons inherently complex. The figure mirrors the structure of Figure 2, reporting accuracy, F1 score, parameters, FLOPs, and maximum tensor size. The proposed model is annotated with its standard deviation over ten independent runs. It is also the only configuration that fits within both STM32F7 and F401RE constraints, as denoted by the ✓✓ symbol; a cross (×) marks configurations where deployment is not feasible.

TABLE VI
ACCURACY AND HARDWARE FOR MULTIPLE EXPERIMENTS

Meth.	Input	VPN-D. (%)	VPN-T. (%)	NVPN-T. (%)	T-Cat. (%)	Par. (M)	M-Tens. (K)	FLOPs (M)
Prop.	**Sess.**	**99.86 ±0.08**	**99.14 ±0.17**	**94.04 ±0.70**	**96.74 ±0.57**	**0.09**	**20.12**	**10.08**
[24]	Sess.	100	99.00	98.70	98.20	39.12	151.51	251.06
[11]	Sess.	99.90	98.30	81.70	-	5.83	25.09	39.73
[5]	Hybrid	99.82	99.38	99.45	-	138.45	4500.00	153.87
[18]	Flows	99.97	94.80	89.30	-	1.35	2.00	0.86
[18]	Flows	99.50	92.90	85.10	-	39.49	2.56	0.67

Packet-level methods, such as the SAE of [6], proved efficient, requiring 0.81M parameters and 1.62M FLOPs in their most efficient setup. However, its parameter count still exceeds the available flash memory of our target MCUs unless structural reduction is applied. This setup achieves an F1 score of 92.00%. The CNN-based approach of [6] is a larger model, demanding 3.55M parameters and 208.19M FLOPs (F1 score 93.00%). Similarly, [51] incurs high hardware costs, requiring 0.69M parameters and 259.26M FLOPs.

Flow-based methods [18], which process flows as time series with predefined packet counts and lengths, show similar trade-offs. While their configurations are relatively efficient in terms of FLOPs, their parameter counts (1.35M and 39.49M, respectively) surpass the limits of the selected target devices. Their reported accuracy (91.20% and 89.50%, respectively) remains lower than that of the proposed model (96.60%). Similarly, [25], which focuses on flow headers for classification, achieves an accuracy of 99.20% but requires 5.77M parameters and 400.43M FLOPs, exceeding the target resource envelope.

Hybrid input methods, like [5], which combines raw packet data with hand-crafted session features, and [23], which segments sessions based on thresholds, achieve superior accuracy (99.70% and 99.10%, respectively). These methods have high requirements, with [5] demanding 138.45M parameters, 153.87M FLOPs, and a max tensor size of 4500.00K, and [23] requiring 6.80M parameters and 401.89M FLOPs.

This analysis highlights the balance of competitive accuracy and remarkable efficiency achieved by the proposed model, solidifying its position as a reference point for hardware-aware TC, even when compared across different problem formulations and input setups. While some other methods report higher accuracy, they do so at the cost of exceeding the memory or compute constraints set by the target devices selected in this work.

C. Evaluation on Multiple Traffic Classification Tasks

Table VI includes accuracy and hardware efficiency metrics for different methods in diverse TC tasks. Each row represents a method (Meth.) and its corresponding input type, while columns detail accuracy for VPN-Diff (VPN-D.), VPN-Type (VPN-T.), NonVPN-Type (NVPN-T.), and Traffic-Cat (T-Cat.) tasks, along with parameters (Par.), max tensor size (M-Tens.), and FLOPs as hardware measures. Proposed model results are reported as mean ± standard deviation over ten runs.

The proposed model (Prop.) achieves near-perfect accuracy (99.86%) in VPN-Diff, matching [24] and surpassing [5] (99.82%). In VPN-Type classification, it achieves 99.14%, outperforming [18] (94.80% and 92.90%) and [11] (98.30%).

TABLE VII
PERFORMANCE FOR THE APP-ID EXPERIMENT

Meth.	Input	Acc. (%)	F1 (%)	Par. (M)	M-Tens. (K)	FLOPs (M)
Prop.	**Sess.**	**94.18 ±0.54**	**94.42 ±0.51**	**0.09**	**20.12**	**10.08**
[5]	Hybrid	99.67	99.51	138.45	4500.00	153.87
[6]	Packets	98.00	98.00	3.55	100.00	208.19
[6]	Packets	-	95.00	0.81	0.40	1.62

In the NonVPN-Type task, the model delivers strong performance (94.04%), exceeding [18] (89.30% and 85.10%) and [11] (81.70%). For Traffic-Cat, a challenging task addressed only by [24], the proposal scores 96.74%. Although [24] obtains slightly higher accuracy (98.20%), the proposal maintains competitive performance while being resource-efficient.

The results validate the proposal across a broad spectrum of tasks, achieving high accuracy and often outperforming SOTA models. As the architecture was optimized for the VPN-NonVPN classification task using HW-NAS, the proposed results can be considered a worst-case analysis.

D. Performance on Application Identification (App-ID)

The App-ID experiment assesses the ability of the proposed session-level model to classify specific applications, a task traditionally dominated by packet-level approaches and hybrid methods [5]. Table VII presents accuracy (Acc.), F1 score (F1), and hardware efficiency metrics: parameters (Par.), maximum tensor size (M-Tens.), and FLOPs (M) for the App-ID experiment. Each row represents a method (Meth.) and its corresponding input type. Scores for our model include standard deviation across ten independent runs.

The proposal (Prop.) achieves 94.18% accuracy and 94.42% F1 score with reduced hardware cost. [6] presents two packet-level models: one with higher accuracy (98.00%) but greater resource use, and another reaching 95.00% F1 with 0.81M parameters and 1.62M FLOPs. The latter model would require a Flash memory larger than the one available in the target devices. Packet-level models operate at finer granularity, which leads to larger training volumes and more frequent inference. While this may enable fast per-packet predictions and strong App-ID accuracy, it also results in higher cumulative processing overhead. These structural differences inherently affect both performance and resource use, and should be considered when interpreting the comparison with session-level methods.

CBS [5] achieves the highest accuracy (99.67%) and F1 score (99.51%) but requires higher resource consumption. Moreover, it was explicitly designed for offline use and is documented as unsuitable for real-time scenarios, which limits its applicability to embedded deployment.

These results confirm the feasibility of session-level models for application classification, achieving competitive performance with substantially lower hardware cost. Despite the inherent granularity differences, the session-level approach remains a practical and scalable option for constrained platforms.

E. Cross-Dataset Evaluation

This subsection evaluates how well the architecture discovered by HW-NAS on ISCX generalizes to other traffic distributions. We report results on USTC-TFC2016 and QUIC NetFlow. The proposed model is re-evaluated on these datasets using the same architecture found during search (see Table V).

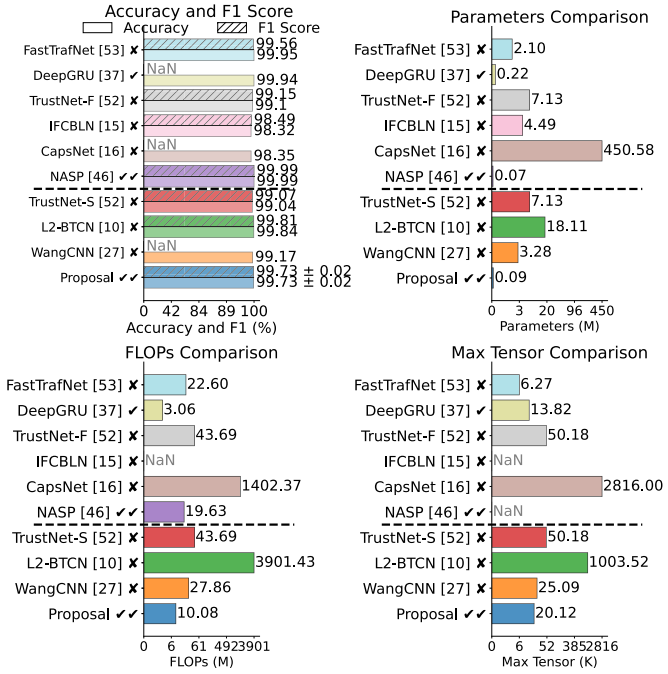

Fig. 4. Results on the USTC-TFC2016 dataset.

Figure 4 shows results on the USTC-TFC2016 dataset. The comparison includes session-based models [10], [27], [52], flow-based models [15], [16], [46], [52], and packet-based models [37], [53]. A horizontal line in the figure separates the session-based approaches (shown below), which are the focus of this study, from the rest. Reported scores for the proposed model include standard deviation over ten independent runs. Device compatibility follows the same notation as previous figures: ✓ for STM32F7, ✓✓ for both STM32F7 and F401RE, and × where deployment is not feasible.

The proposed model achieves 99.73% accuracy and 99.73% F1 score with 0.09M parameters, 10.08M FLOPs, and a 20.12K max tensor. Although optimized on ISCX, it maintains strong performance on USTC with minimal hardware cost.

Among session-level models, [10] reports the highest accuracy (99.84%), but at the cost of 18.11M parameters and over 3900M FLOPs. The session-based variant of [52] also incurs high compute demand (7.13M parameters, 43.69M FLOPs). [27] achieves similar accuracy (99.17%) with fewer resources than the others in its category, though its compute and memory footprint remains higher than ours. None of these baselines are deployable on the tested MCUs due to excessive parameter and tensor sizes, making our model the only viable session-level solution in constrained settings.

The NAS-based flow model from [46], tailored for USTC, reaches 99.99% accuracy with slightly fewer parameters (0.07M) but nearly doubles the FLOPs (19.63M). Its max tensor is unreported, but we conservatively assume it fits within memory limits. However, its higher compute load suggests greater energy use and latency than ours on similar hardware. Other flow-level models [15], [16], [52] are far more complex, with parameter counts from 4.49M to 450.58M.

TABLE VIII
QUIC NETFLOW: CLASSIFICATION PERFORMANCE

Method	Input	Acc. (%)	F1 (%)
Proposal	**Sess.**	**99.98 ±0.01**	**99.98 ±0.01**
CNN + RF [28]	Hybrid	99.00	99.00
DWT-TC [54]	Features	93.70	93.60
STFT-TC [54]	Features	95.50	95.40
RF [32]	Features	95.30	95.30

The model presented in [37] achieves 99.94% accuracy with 0.22M parameters, 3.06M FLOPs, and a 13.82K tensor, making it compatible with STM32F7-class devices. However, it exceeds the Flash constraints of F401RE and operates at packet-level granularity, requiring inference at finer time scales. This incurs higher cumulative overhead than our session-level model, which classifies entire sessions in a single pass. [53] combines high accuracy (99.95%) with the smallest tensor (6.27K), yet its 2.10M parameter count far exceeds the Flash capacity of our target devices.

Table VIII presents classification results on the QUIC NetFlow dataset. Each row corresponds to a method, while columns report input type, accuracy, and F1 score. Hardware metrics for existing methods are omitted due to missing implementation details. Compared approaches rely on flow-level features [32], handcrafted signals [54], or hybrid pipelines combining raw bytes with flow features [28].

Our model attains the highest accuracy (99.98%) and F1 score (99.98%) with minimal complexity (0.09M parameters, 10.08M FLOPs); results are averaged over ten runs. CNN+RF [28] performs competitively (99.00%) but uses a two-stage pipeline with limited deployment suitability. Feature-based methods [32], [54] lag by 4 to 6 points. These results show that a compact session-based model using raw bytes can outperform protocol-specific and feature-heavy pipelines, even in specialized QUIC traffic scenarios.

Overall, these findings confirm the model's ability to generalize across encrypted traffic datasets with differing characteristics, while maintaining its efficiency advantage.

VII. EVALUATION AND ANALYSIS

A. Preprocessing Analysis

Header-level preprocessing is critical for balancing privacy and the retention of key structural features essential for TC. Packet headers carry structural and protocol information that is particularly relevant for ETC, where the payload is obfuscated and contributes little to feature extraction. This study modifies specific header fields, commonly targeted in prior works, to evaluate their impact on classification performance in the VPN-NonVPN task from ISCX, which serves as the reference benchmark for analysis. The dataset predominantly includes TCP and UDP protocols, where headers follow a layered hierarchy: the Ethernet layer (14 bytes), containing source and destination MAC addresses (6 bytes each) and EtherType (2 bytes); the IP layer, which includes source and destination IP addresses (4 bytes each); and the transport layer, which adds source and destination ports (2 bytes each) and protocol headers, with UDP headers being 8 bytes and TCP headers 20 bytes. These modifiable fields collectively account for approximately 26 to 38 bytes per packet, out of the standardized session input size of 784 bytes.

TABLE IX
EFFECT OF PREPROCESSING ON MODEL PERFORMANCE

Strat.	Eth. Rem.	MAC Anon.	MAC Zero	IP Anon.	IP Zero	Port Zero	UDP Pad.	Acc. (%)
1	✓			✓			✓	95.35
2	✓			✓				96.59
3	✓			✓		✓	✓	95.80
4	✓			✓		✓		96.12
5	✓				✓		✓	95.56
6	✓				✓			95.38
7	✓				✓	✓	✓	95.12
8	✓				✓	✓		95.00
9		✓		✓			✓	94.79
10		✓		✓				95.42
11		✓		✓		✓	✓	95.61
12		✓		✓		✓		95.82
13		✓			✓		✓	95.45
14		✓			✓			94.70
15		✓			✓	✓	✓	94.86
16		✓			✓	✓		93.42
17			✓	✓			✓	96.19
18			✓	✓				95.59
19			✓	✓		✓	✓	95.66
20			✓	✓		✓		96.08
21			✓		✓		✓	94.30
22			✓		✓			92.25
23			✓		✓	✓	✓	95.24
24			✓		✓	✓		89.08

Table IX evaluates the effects of 24 preprocessing strategies (Strat.), i.e., all possible combinations of preprocessing choices for header fields, on the model derived from the NAS. Each row represents a strategy, with columns describing applied steps such as removing the Ethernet layer (Eth. Rem.), anonymizing (Anon.; replacing field values with hashed pseudonyms), zeroing (Zero; setting values to 0) MAC and IP addresses, zeroing protocol-related fields like ports (Port Zero), and applying UDP padding (UDP Pad.) to align UDP and TCP segments. A checkmark (✓) indicates that a step was applied. The final column shows the accuracy (%; Acc.). It should be noted that zeroing enhances privacy by replacing original values with zeros, which removes meaningful variability and makes it harder for the model to infer patterns.

Strategy 2, used during the NAS process, achieves the highest accuracy (96.59%) and represents a frequently adopted approach in prior works. It anonymizes IP addresses without zeroing them and avoids UDP padding, retaining key features for classification. The model also performs well under more restrictive preprocessing, like strategy 6 (95.38%), which zeros IP addresses, or strategy 7 (95.12%), which zeroes both IP addresses and ports while applying UDP padding. These results show that the proposed model generalizes well, without overfitting specific header fields like IP addresses or ports.

Some strategies cause drops in accuracy when compared to Strategy 2. For example, Strategy 24, which zeros MAC addresses, IP addresses, and ports, leads to the lowest accuracy (89.08%), with a drop of 7.51%. Strategy 22, similar to Strategy 24 but without port zeroing, performs slightly better (92.25%) but still exhibits a 4.34% drop, suggesting that excessive zeroing disrupts critical structural patterns. However, Strategies 23 and 21 (95.24% and 94.30%, respectively), using

TABLE X
IMPACT OF INPUT SIZE ON HARDWARE METRICS

Input Size (B)	784	676	576	484	400	324	256	196
FLOPs (M)	10.08	8.61	7.25	6.07	4.90	3.95	3.01	2.24
M-Tens. (K)	20.12	17.29	14.71	12.38	10.19	8.26	6.45	4.90

UDP padding, absent in Strategies 22 and 24, partially mitigate the accuracy drops caused by extensive zeroing.

Removing the Ethernet layer entirely, as in Strategy 8 (95.00%), outperforms zeroing MAC addresses in Strategy 24 (89.08%). Both strategies are identical in other preprocessing steps. This complete removal prevents partial masking and preserves structural consistency, yielding better performance. We discourage anonymizing MAC addresses, as demonstrated in Strategies 9 through 16. For example, Strategy 10 (95.42%) takes a moderate approach by anonymizing both MAC and IP addresses instead of zeroing them but still yields limited gains. It is 1.17% below the best strategy, highlighting the minimal impact MAC addresses have on classification.

The proposal performs robustly across many configurations. Strategies like 7 and 23 demonstrate its adaptability, achieving a commendable balance between privacy preservation and classification performance, making it suitable for applications with privacy regulations. These findings help practitioners strike the right balance between obfuscating sensitive fields and keeping essential header features for optimal performance.

B. Impact of Session Length on Performance

This analysis tests the impact of session length on the trade-offs between efficiency and classification performance in the VPN-NonVPN task. Header-level strategies affect accuracy, leaving hardware measures unchanged, while session length reductions impact efficiency by altering the input size. We considered the first eight preprocessing strategies in Table IX because the Ethernet layer is removed, a common practice in TC. We progressively reduced the standardized session length from 784 bytes (the standard value used in most research papers) to smaller lengths: 676, 576, 484, 400, 324, 256, and 196 bytes. The reductions are selected to capture meaningful performance trends while preserving key session-level features, such as critical header fields and early-packet information essential for classification.

Table X shows how different input sizes, listed in the columns, affect FLOPs and maximum tensor size (M-Tens.), listed in the rows, regardless of preprocessing strategy. As input size decreases, FLOPs drop from 10.08M to 2.24M, and M-Tens. reduces proportionally from 20.12K to 4.90K. Smaller input sizes improve efficiency, making the model more suitable for deployment in resource-constrained environments. The number of parameters remains constant at 88.26K.

Figure 5 complements Table X by showing how accuracy varies with input size across the first eight strategies from Table IX. The x-axis represents input size, while the y-axis shows accuracy for each strategy. The figure highlights that for most strategies (1, 2, 3, 5, 6, 7), a small drop of 1 to 2% at 196 bytes affected the accuracy. For strategies with extensive zeroing, such as Strategy 8 (zeros IP addresses and ports) and Strategy 4 (zeros ports but anonymizes IP addresses), the accuracy drop becomes more pronounced, reaching up to 4% as input size decreases. Conversely, Strategies 7 and 3, which follow the same steps as 8 and 4, respectively, but include UDP padding, mitigate this issue. Incorporating padding mechanisms can trade off input size reduction and classification performance, as shown by the table and figure.

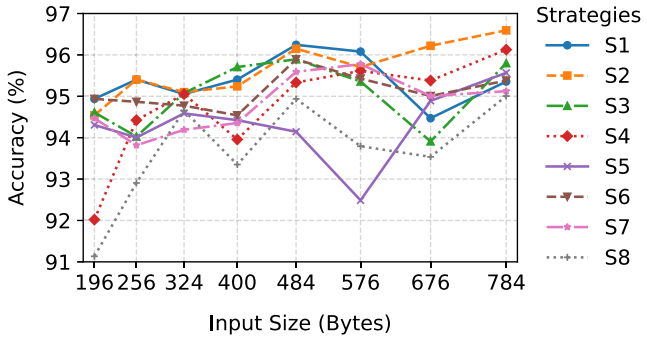

Fig. 5. Accuracy across strategies for different input sizes.

VIII. EMBEDDED INFERENCE DEPLOYMENT

A. Deployment Methodology and Setup

Building on the session length analysis in Section VII-B, we evaluate how the model selected by HW-NAS performs on MCUs across various input sizes. All deployment results use this architecture and preprocessing Strategy 2 from Table IX, which served as the main configuration throughout this study.

We consider a deployment setting in which fixed-length session vectors are prepared on an upstream device (e.g., an IoT gateway or embedded Linux board), which is assumed to perform packet capture, session aggregation, and buffering, and forwarded to a microcontroller (e.g., STM32F401RE or STM32F746G), which executes the classification stage under tight memory and compute constraints. Inference is triggered when the required input representation is available, and real-time operation is therefore defined in terms of bounded response time at the session level. Our evaluation focuses on MCU-side execution, which represents the actual resource-constrained component of the deployment pipeline and determines the feasibility of on-device TC.

The HW-NAS-optimized architecture was quantized to 8-bit (INT8) using TensorFlow Lite and converted to C code via STM32Cube.AI. Both post-training quantization (PTQ) and quantization-aware training (QAT) were considered; PTQ applies INT8 conversion to a pretrained Float32 model, while QAT fine-tunes the pretrained model under simulated INT8 arithmetic to mitigate quantization-induced accuracy loss. For each input size, PTQ and QAT were repeated 10 times to evaluate accuracy variability; one representative INT8 model per input size was deployed, as all quantized variants share the same architecture, memory footprint, and computational cost.

Inference metrics were averaged over 1000 runs per device. A USB power meter, connected between the board and host PC, recorded current and voltage during idle and active phases. Power per session was derived from the average current and voltage during the active phase, and energy was calculated as $E = P \cdot t$, where t is the inference latency.

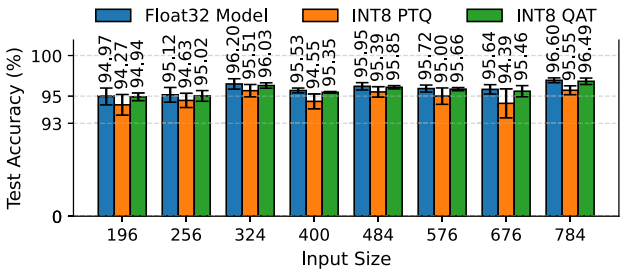

Fig. 6. Accuracy across quantization levels (VPN-NonVPN).

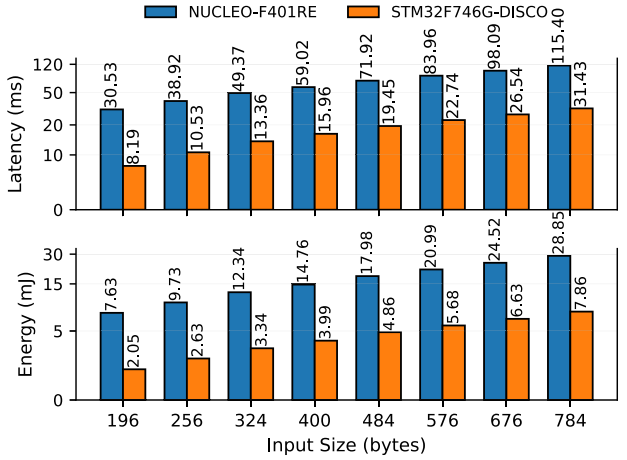

Fig. 7. Inference cost across MCUs and input sizes.

B. Experimental Results and Discussion

Figure 6 compares model accuracy across Float32 and INT8 configurations (PTQ and QAT) for input sizes from 196 to 784 bytes. The x-axis reports the input size, while the y-axis shows test accuracy (%). Error bars indicate the standard deviation across 10 independent training and quantization runs.

INT8-PTQ incurs a moderate but consistent accuracy degradation; QAT recovers most of this loss and significantly reduces variance across runs. Across all input sizes, QAT remains within 0.1–0.3% of the Float32 baseline, confirming stable and deployment-ready behavior.

Figure 7 illustrates inference latency (top, in ms) and energy per session (bottom, in mJ) across input sizes and MCUs.

During inference, both boards exhibited an average current draw of 0.05 A at a 5 V supply, resulting in an average active power of 0.25 W across the tested configurations. Idle current was measured separately: 0.02 A for the F401RE and 0.23 A for the F746G, corresponding to idle power baselines of 0.10 W and 1.15 W, respectively. Energy values in Figure 7 are based on active power only, isolating the model execution cost from background consumption.

Latency increases with input size, as expected. The Nucleo-F401RE maintains latency below 60 ms for sizes up to 400 bytes, but reaches 115 ms at 784 bytes. The STM32F746G achieves consistently lower latency, ranging from 8 ms to 31 ms, reflecting its faster clock and memory bandwidth. Energy trends follow latency linearly due to constant power.

TABLE XI
INFERENCE LATENCY BY PLATFORM AND POWER BUDGET

Work	Platform	Hardware Class	Power Budget (W)	Input	Latency
Our work	STM32F746G	Cortex-M7	< 1	Session	31.43 ms (784 B)
					19.45 ms (484 B)
	STM32F401RE	Cortex-M4			115.40 ms (784 B)
					71.92 ms (484 B)
[48]	STM32F746G	Cortex-M7	< 1	Session	31.43 ms
	STM32F401RE	Cortex-M4			115.40 ms
[2]	PCIe TPU	Edge accelerator	∼30	Flows	10–30 ms
[32]	Intel Tofino	Switch ASIC	∼20–30	Flows	$< 1\ \mu s^*$

At an input size of 784 bytes, the model runs at its full hardware configuration found by HW-NAS: 88.26K parameters, 20.12K max tensor, and 10.08M operations (the INT8 equivalent of the baseline FLOPs), matching the setup used throughout Section VI. This setting yields 115.4 ms latency and 28.85 mJ energy on Cortex-M4, and 31.43 ms and 7.86 mJ on Cortex-M7. These values can serve as an empirical deployment reference for models with similar or higher complexity, assuming deployment on comparable devices.

Overall, the results confirm that HW-NAS models can be deployed on low-power MCUs for real-time TC. Input sizes like 324 and 484 bytes offer a good trade-off: they achieve over 95% INT8 accuracy while keeping latency under 72 ms on Cortex-M4 and 20 ms on Cortex-M7; energy remains below 18 mJ and 5 mJ, respectively. Smaller inputs such as 256 bytes are more efficient but slightly less accurate. Despite the increase in latency with input size, delays remain acceptable in realistic scenarios, especially given that session aggregation and preprocessing occur on the gateway. These values align with response time ranges in embedded monitoring tasks [49].

The literature includes only a few works that report deployment measurements, and the target platform varies across studies. Table XI summarizes inference latency reported in prior work. Each row corresponds to a method and platform, while the columns report the hardware class, power budget (in watts), input type, and inference latency. For our work, two session input sizes are reported per platform, corresponding to a maximum-length configuration (784 B) and a shorter operating point (484 B), where B denotes bytes. Underlining highlights the best latency among low-power deployments (power budget <1 W), which is the primary target scenario of this work, while the asterisk (*) denotes the best overall latency across all platforms.

By comparison, STM32-based inference was previously shown in [48], but only for a single input size (784 bytes), without examining empirical trade-offs across different input lengths. FENXI [2] reports 10–30 ms inference latency using a high-power (30 W) tensor processing unit (TPU) attached via PCI Express, under 100 Gbps traffic with batching and model caching. The in-switch approach of [32] achieves sub-microsecond latency at line rate by executing Random Forest models directly on Intel Tofino programmable switches; however, this relies on high-end network hardware, and the feature pipeline is limited by the P4 programming model, which restricts flexibility and applicability to broader embedded settings. In contrast, our results show that ETC inference is feasible on microcontrollers with a power budget below 1 W,

and that practical latency can be achieved by varying the input length to balance efficiency and response time.

IX. Conclusion and Future Work

This paper introduces a DNN optimized through HW-NAS for efficient TC in resource-constrained environments. The study shows that incorporating hardware constraints during architecture synthesis supports reproducible and deployable design choices, where accuracy is achieved under strict efficiency requirements. The proposed model achieves up to 96.60% accuracy in session-level classification while requiring up to 444 times fewer parameters and 312 times fewer FLOPs than SOTA methods. These results highlight its suitability for IoT and edge platforms, where efficiency is critical.

The optimized architecture delivers satisfactory performance across diverse tasks and settings; it also obtains consistent performance in terms of cross-dataset generalization, confirming its adaptability to different TC challenges. The study further shows that reducing input size improves efficiency without sacrificing performance; combined with quantization, this yields notable latency and energy gains on embedded microcontrollers. Additionally, techniques like UDP padding mitigate accuracy loss in restrictive preprocessing scenarios, offering practical solutions.

Future work will explore broader deployment scenarios, including adaptive runtime optimization and integration with real-time network operations. Federated learning will also be investigated to enable decentralized training with privacy preservation and improved scalability. We further plan to extend HW-NAS to other traffic analysis tasks, such as intrusion and anomaly detection, to assess its adaptability across domains, and to incorporate additional deployment-aware objectives, such as quantization-aware constraints and robustness considerations (e.g., adversarial perturbations and distribution shift), thereby improving reliability under practical operating conditions. These directions aim to advance hardware-efficient ML for next-generation network systems.

References

[1] M. Martalò, G. Pettorru, and L. Atzori, "A cross-layer survey on secure and low-latency communications in next-generation IoT," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 4, pp. 4669–4685, Aug. 2024.

[2] M. Gallo, A. Finamore, G. Simon, and D. Rossi, "FENXI: Deep-learning traffic analytics at the edge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Dec. 2021, pp. 202–213.

[3] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1988–2014, 2nd Quart., 2019.

[4] W. Dong, J. Yu, X. Lin, G. Gou, and G. Xiong, "Deep learning and pre-training technology for encrypted traffic classification: A comprehensive review," *Neurocomputing*, vol. 617, Feb. 2025, Art. no. 128444.

[5] M. Seydali, F. Khunjush, B. Akbari, and J. Dogani, "CBS: A deep learning approach for encrypted traffic classification with mixed spatio-temporal and statistical features," *IEEE Access*, vol. 11, pp. 141674–141702, 2023.

[6] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Comput.*, vol. 24, no. 3, pp. 1999–2012, Feb. 2020.

[7] M. Adil, H. Song, M. K. Khan, A. Farouk, and Z. Jin, "5G/6G-enabled metaverse technologies: Taxonomy, applications, and open security challenges with future research directions," *J. Netw. Comput. Appl.*, vol. 223, Mar. 2024, Art. no. 103828.

[8] N. Malekghaini et al., "AutoML4ETC: Automated neural architecture search for real-world encrypted traffic classification," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 3, pp. 2715–2730, Jun. 2024.

[9] E. Tabanelli, G. Tagliavini, and L. Benini, "DNN is not all you need: Parallelizing non-neural ML algorithms on ultra-low-power IoT processors," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 3, pp. 1–33, May 2023.

[10] Z. Li and X. Xu, "L2-BiTCN-CNN: Spatio-temporal features fusion-based multi-classification model for various internet applications identification," *Comput. Netw.*, vol. 243, Apr. 2024, Art. no. 110298.

[11] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proc. IEEE Int. Conf. Intell. Secur. Informat. (ISI)*, Jul. 2017, pp. 43–48.

[12] W. Maonan, Z. Kangfeng, X. Ning, Y. Yanqing, and W. Xiujuan, "CENTIME: A direct comprehensive traffic features extraction for encrypted traffic classification," in *Proc. IEEE 6th Int. Conf. Comput. Commun. Syst. (ICCCS)*, Apr. 2021, pp. 490–498.

[13] F. Hu, S. Zhang, X. Lin, L. Wu, N. Liao, and Y. Song, "Network traffic classification model based on attention mechanism and spatiotemporal features," *EURASIP J. Inf. Secur.*, vol. 2023, no. 1, p. 6, Jul. 2023.

[14] A. Zou, W. Yang, C. Tang, J. Lu, and J. Guo, "A novel and effective encrypted traffic classification method based on channel attention and deformable convolution," *Comput. Electr. Eng.*, vol. 118, Aug. 2024, Art. no. 109406.

[15] J. Xu et al., "A cascaded broad learning network embedded image features for malware traffic classification," *IEEE Trans. Cognit. Commun. Netw.*, vol. 11, no. 4, pp. 2426–2439, Aug. 2025.

[16] H. Yao, P. Gao, J. Wang, P. Zhang, C. Jiang, and Z. Han, "Capsule network assisted IoT traffic classification mechanism for smart cities," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7515–7525, Oct. 2019.

[17] Z. Bu, B. Zhou, P. Cheng, K. Zhang, and Z.-H. Ling, "Encrypted network traffic classification using deep and parallel network-in-network models," *IEEE Access*, vol. 8, pp. 132950–132959, 2020.

[18] H. Yao, C. Liu, P. Zhang, S. Wu, C. Jiang, and S. Yu, "Identification of encrypted traffic through attention mechanism based long short term memory," *IEEE Trans. Big Data*, vol. 8, no. 1, pp. 241–252, Feb. 2022.

[19] C. White et al., "Neural architecture search: Insights from 1000 papers," 2023, *arXiv:2301.08727*.

[20] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, and A. K. Somani, "Neural architecture search benchmarks: Insights and survey," *IEEE Access*, vol. 11, pp. 25217–25236, 2023.

[21] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 2820–2828.

[22] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.

[23] S. Cui, B. Jiang, Z. Cai, Z. Lu, S. Liu, and J. Liu, "A session-packets-based encrypted traffic classification using capsule neural networks," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun., IEEE 17th Int. Conf. Smart City, IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 429–436.

[24] B. Lu, N. Luktarhan, C. Ding, and W. Zhang, "ICLSTM: Encrypted traffic service identification based on inception-LSTM neural network," *Symmetry*, vol. 13, no. 6, p. 1080, Jun. 2021.

[25] S. Cui, J. Liu, C. Dong, Z. Lu, and D. Du, "Only header: A reliable encrypted traffic classification framework without privacy risk," *Soft Comput.*, vol. 26, no. 24, pp. 13391–13403, Dec. 2022.

[26] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related," in *Proc. 2nd Int. Conf. Inf. Syst. Security Privacy (ICISSP)*, 2016, pp. 407–414.

[27] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2017, pp. 712–717.

[28] V. Tong, H. A. Tran, S. Souihi, and A. Mellouk, "A novel QUIC traffic classifier based on convolutional neural networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[29] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *Proc. 13th Int. Conf. World Wide Web*, May 2004, pp. 512–521.

[30] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. Int. Workshop Passive Active Netw. Meas.*, 2005, pp. 41–54.

[31] Y. Zhai and X. Zheng, "Random forest based traffic classification method in SDN," in *Proc. Int. Conf. Cloud Comput., Big Data Blockchain (ICCBB)*, Nov. 2018, pp. 1–5.

[32] A. T.-J. Akem, G. Fraysse, and M. Fiore, "Encrypted traffic classification at line rate in programmable switches with machine learning," in *Proc. NOMS - IEEE Netw. Oper. Manage. Symp.*, May 2024, pp. 1–9.
[33] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. V. Vasilakos, "An effective network traffic classification method with unknown flow detection," *IEEE Trans. Netw. Service Manage.*, vol. 10, no. 2, pp. 133–147, Jun. 2013.
[34] Y. He and W. Li, "Image-based encrypted traffic classification with convolution neural networks," in *Proc. IEEE 5th Int. Conf. Data Sci. Cyberspace (DSC)*, Jul. 2020, pp. 271–278.
[35] M. Wang, K. Zheng, D. Luo, Y. Yang, and X. Wang, "An encrypted traffic classification framework based on convolutional neural networks and stacked autoencoders," in *Proc. IEEE 6th Int. Conf. Comput. Commun. (ICCC)*, Dec. 2020, pp. 634–641.
[36] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in SDN home gateway," *IEEE Access*, vol. 6, pp. 55380–55391, 2018.
[37] B. Wang, Y. Su, M. Zhang, and J. Nie, "A deep hierarchical network for packet-level malicious traffic detection," *IEEE Access*, vol. 8, pp. 201728–201740, 2020.
[38] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012.
[39] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, 2018, pp. 108–116.
[40] M. Basit Umair, Z. Iqbal, M. Bilal, T. Adnan Almohamad, J. Nebhen, and R. Majid Mehmood, "An efficient internet traffic classification system using deep learning for IoT," 2021, *arXiv:2107.12193*.
[41] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "A comprehensive survey on hardware-aware neural architecture search," 2021, *arXiv:2101.09336*.
[42] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 11711–11722.
[43] E. Ragusa, F. Zonzini, P. Gastaldo, and L. De Marchi, "Combining compressed sensing and neural architecture search for sensor-near vibration diagnostics," *IEEE Trans. Ind. Informat.*, vol. 20, no. 8, pp. 10488–10498, Aug. 2024.
[44] E. Ragusa, F. Zonzini, L. De Marchi, and R. Zunino, "Compression–accuracy co-optimization through hardware-aware neural architecture search for vibration damage detection," *IEEE Internet Things J.*, vol. 11, no. 19, pp. 31745–31757, Oct. 2024.
[45] A. Mattia Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "Searching neural architectures for sensor nodes on IoT gateways," 2025, *arXiv:2505.23939*.
[46] X. Zhang, L. Hao, G. Gui, Y. Wang, B. Adebisi, and H. Sari, "An automatic and efficient malware traffic classification method for secure Internet of Things," *IEEE Internet Things J.*, vol. 11, no. 5, pp. 8448–8458, Mar. 2024.
[47] A. Chehade, E. Ragusa, P. Gastaldo, and R. Zunino, "Tiny neural networks for session-level traffic classification," in *Proc. Int. Conf. Appl. Electron. Pervading Ind., Environ. Soc.*, 2024, pp. 347–354.
[48] A. Chehade, E. Ragusa, P. Gastaldo, and R. Zunino, "Energy-efficient deep learning for traffic classification on microcontrollers," 2025, *arXiv:2506.10851*.
[49] X. Jiang et al., "Low-latency networking: Where latency lurks and how to tame it," *Proc. IEEE*, vol. 107, no. 2, pp. 280–306, Feb. 2019.
[50] M. Song, J. Ran, and S. Li, "Encrypted traffic classification based on text convolution neural networks," in *Proc. IEEE 7th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Oct. 2019, pp. 432–436.
[51] Y. Zhou et al., "Identification of encrypted and malicious network traffic based on one-dimensional convolutional neural network," *J. Cloud Comput.*, vol. 12, no. 1, p. 53, Apr. 2023.
[52] Z. Li, Y. Liu, C. Zhang, W. Shan, H. Zhang, and X. Zhu, "Trustworthy deep learning for encrypted traffic classification," *Soft Comput.*, vol. 29, no. 2, pp. 645–662, Jan. 2025.
[53] L. Yu, J. Yuan, J. Zheng, and N. Yang, "A model of encrypted network traffic classification that trades off accuracy and efficiency," *J. Netw. Syst. Manage.*, vol. 33, no. 1, pp. 1–32, Jan. 2025.
[54] N. Dillbary, R. Yozevitch, A. Dvir, R. Dubin, and C. Hajaj, "Hidden in time, revealed in frequency: Spectral features and multiresolution analysis for encrypted internet traffic classification," in *Proc. IEEE 21st Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2024, pp. 266–271.

Adel Chehade (Graduate Student Member, IEEE) received the B.S. degree in electronics and the M.S. degree in signal, telecoms, image, and speech from Lebanese University, Lebanon, in 2021 and 2023, respectively. He is currently pursuing the Ph.D. degree with the Department of Naval, Electrical, Electronic and Telecommunications Engineering (DITEN), University of Genoa, Italy. His research interests include hardware-aware machine learning and embedded systems, with a focus on TinyML applications for network security and edge intelligence.

Edoardo Ragusa (Member, IEEE) received the master's degree (cum laude) in electronic engineering and the Ph.D. degree in electronic engineering from the University of Genoa, Genoa, Italy, in 2015 and 2018, respectively. He is currently a Tenure Track Researcher with DITEN, University of Genoa, where he teaches digital systems electronics and tiny machine learning. He has co-authored more than 70 refereed papers in international journals and conferences. His research interests include machine learning in resource-constrained devices, convolutional neural networks, and deep learning applications.

Paolo Gastaldo (Member, IEEE) received the Laurea degree in electronic engineering and the Ph.D. degree in space sciences and engineering from the University of Genoa, Italy, in 1998 and 2004, respectively. He is currently an Associate Professor with the Department of Naval, Electrical, Electronic and Telecommunications Engineering (DITEN), University of Genoa, where he teaches computer architecture and sensors. His research focuses on embedded machine learning and intelligent embedded systems, with applications to robotics, prosthetics, structural health monitoring, and cybersecurity. He has authored over 100 peer-reviewed publications in international journals and conferences.

Rodolfo Zunino was born in Genoa, Italy, in 1961. He received the Laurea degree (cum laude) in electronic engineering from Genoa University in 1985. From 1986 to 1995, he was a Research Consultant with the Department of Biophysical and Electronic Engineering (DIBE, now DITEN), Genoa University. He is currently with DITEN as a Full Professor in electronics and electronics for embedded systems. He is also the Head of the Smart Embedded Applications Laboratory (SEALab), DITEN, and also the Head of the Master in Cyber Security and Critical Infrastructure Protection, Genoa University. His main scientific research interests include embedded electronic systems for neural networks, efficient models for data representation and learning, massive-scale text-mining and text-clustering methods, advanced techniques for multimedia data processing, intelligent systems for computer security, network security, and critical infrastructure protection. He is the co-inventor of two patents and co-authored more than 270 scientific papers in international journals and conferences. He has been the Co-Chair of the two editions of the International Workshop on Computational Intelligence for Security in Information Systems (CISIS'08 and CISIS'09). From 2001 to 2010, he contributed as an Associate Editor of IEEE TRANSACTIONS ON NEURAL NETWORKS and participates in the scientific committees of several international events.

Open Access funding provided by 'Universita degli studi di Genova' within the CRUI CARE Agreement