EAI Endorsed Transactions
on Industrial Networks and Intelligent Systems
Research Article EAI.EU

PON: A Packet-Observation Spiking Neural Network for Header-Based IoT Intrusion Detection

Phu Nguyen Phan Hai[1,2], Cam Doan Ngoc[1,2], Bao Bui Quoc[1,2], Trang Hoang[1,2,*]

[1]Faculty of Electrical and Electronics Engineering, Ho Chi Minh City University of Technology, 268 Ly Thuong Kiet Street, Ho Chi Minh City, 700000, Dien Hong Ward, Vietnam
[2]Vietnam National University, Ho Chi Minh City, Linh Xuan Ward, Ho Chi Minh City, 700000, Vietnam

Abstract

Intrusion detection in IoT environments requires real-time and lightweight systems to mitigate the computational overhead of deep packet inspection (DPI) at the network-edge. This paper proposes PON, a packet-observation spiking neural network (SNN) for IoT intrusion detection using packet headers. The system operates online at the individual packet-level, incorporating a behavior-adaptive redundancy filter that reduces edge processing traffic volume by over 79 percent. Header features are constructed from 64-byte normalized headers alongside a sliding-window source IP diversity ratio to capture the behaviors of diverse cyber attacks. PON performs header-only inference using these features, while payload data are not forwarded to the classifier. The PON architecture combines 1D convolution with leaky-integrate-and-fire (LIF) neurons and a Sparse Attack Detector (SAD) designed to mitigate temporal spike dilution, thereby boosting rare attack detection. Evaluated on the mixed traffic streams of the CIC-IoT2023 dataset, the proposed SNN model combined with the SAD module reaches a Macro F1 score of 92.48 percent and a detection rate of 90.23 percent, increasing the detection rate on rare attack categories by up to 21.5 percentage points compared to published reference models. Trained ternary quantization (TTQ) reduces the storage footprint from 30.69 KB in float32 format to 13.82 KB in ternary format, achieving a 2.2 fold overall model size reduction by compressing the core spiking weights 16 fold. These results demonstrate that the proposed model provides competitive detection accuracy and high memory efficiency, making it suitable for edge deployment.

Received on 01 June 2026; accepted on 04 August 2026; published on 19 August 2026

Keywords: IoT intrusion detection, packet-level features, lightweight features, 1D convolution, ternary quantization, CIC-IoT2023

doi:10.4108/eetinis.133.13277

1. Introduction

Internet of things (IoT) devices have become essential components of modern smart environments, spanning domestic, industrial, and healthcare domains [1]. Accordingly, this study focuses on packet-level intrusion detection for IoT devices and evaluates PON using the benign and attack scenarios of CIC-IoT2023. However, these devices typically possess restricted computational capabilities and minimal security protections, making them prime targets for diverse cyber threats. IoT security attacks are no longer restricted to simple denial of service incursions, but comprise multi vector campaigns such as brute force access, malicious firmware spoofing, and stealthy reconnaissance [2, 3]. These diverse network threats can rapidly compromise vulnerable edge devices, leading to data breaches or service disruptions across connected infrastructures, highlighting the vital role of machine learning in cybersecurity defense [4, 5].

To secure IoT networks, intrusion detection systems are conventionally deployed at the edge. Traditional methods rely on flow-based traffic analysis, which aggregates statistical packet features over entire connection sessions. For instance, the method of Wang *et al.* [6] utilizes incremental principal component analysis to reduce the dimensionality of statistical

*Corresponding author. Email: hoangtrang@hcmut.edu.vn

flow features for a lightweight deep neural network model. Similarly, Abbas *et al.* [7] evaluate several deep learning models by extracting network flow features to classify various cyber threats. Although conventional flow features provide a consolidated view, they suffer from a severe architectural limitation because classification can only occur after flow termination, creating a long stalling delay. As highlighted by Djaidja *et al.* [8], waiting for flow termination introduces significant detection latency, which allows malicious traffic to pass. Similarly, Kim and Pak [9] note that relying on completed sessions prevents real-time response, proposing a session independent classifier to detect intrusions early without waiting for session completion. Furthermore, deep learning architectures including convolutional neural networks, long short term memory (LSTM) models, and Transformers have been widely investigated to classify diverse security incursions in IoT environments [10, 11].

To address this detection latency, packet stream based detection has emerged as a promising alternative. Unlike flow-based methods, a packet stream approach processes traffic sequentially at the individual packet-level, enabling real-time online intrusion detection on a continuous basis. Despite this major advantage, packet stream methods introduce unique difficulties. The raw packet stream contains large volumes of redundant traffic, which imposes significant processing and storage overhead on resource-constrained edge hardware [12, 13]. Furthermore, packet-level header features are highly sporadic and sparse, making it challenging to isolate stealthy attack signatures from surrounding normal traffic without expensive DPI. To address these challenges, we introduce a lightweight packet stream preprocessing pipeline before the classification stage. First, a behavior-adaptive redundancy filter is applied to the raw traffic to dynamically compress highly repetitive consecutive packets without state transitions, reducing the traffic volume by over 79 percent and alleviating the processing and storage burden at the edge. Second, to overcome the sporadic nature of packet header attributes, the pipeline constructs a compact feature vector that combines normalized 64-byte header attributes with a sliding-window source IP diversity ratio, capturing the distinct behavioral dynamics of diverse attacks.

SNNs offer an efficient framework for deploying deep learning models on resource-constrained edge hardware, as reviewed in detail by Schuman *et al.* [14] and Roy *et al.* [15]. Because of their event-driven computational efficiency and low-power requirements, SNN architectures have been successfully deployed across a wide range of real-time edge domains. For instance, in neuromorphic vision and fast object recognition, SNNs demonstrate remarkable processing speed with minimal energy consumption [16, 17]. Similarly, in robotics and dynamic motor control, spike-based computation enables real-time autonomous navigation and sensory processing on neuromorphic chips [18]. SNNs are also widely utilized for decoding complex biosignals such as electrocardiograms and electroencephalograms in edge healthcare devices, alongside processing temporal audio streams for lightweight keyword spotting, as discussed by Yamazaki *et al.* [19]. In addition to these cognitive and sensory tasks, SNNs have been extensively researched for securing network systems and protecting IoT infrastructures from cyber threats [20–22]. SNNs emulate biological neural systems by communicating via sparse, discrete binary spikes rather than continuous values, which replaces expensive floating point multiplications with simple conditional additions [19]. This event-driven computational paradigm significantly reduces the energy consumption and memory footprint of edge constrained devices. Nevertheless, SNNs suffer from a temporal spike dilution disadvantage [23, 24]. When sparse attack packets are interleaved with dominant normal traffic within a long temporal window, the membrane potential integration of LIF neurons averages out the anomalous spikes, suppressing the warning signal and leading to a high false-negative rate for rare attacks. To address this potential temporal spike dilution, we propose PON, which combines 1D convolution and LIF neurons and incorporates a behavior-adaptive attention and gating mechanism called the SAD module. The SAD module uses a local window average to detect sudden variations in local spike density, calculating an anomaly score for each timestep relative to the surrounding packet context. This score is used in a multiplicative gating path to dynamically amplify anomalous neuron states and in a sparse attention path to route salient features directly to the classification layer. By selectively enhancing sparse, sporadic attack packets, the detector is designed to prevent critical threat signatures from being averaged out by dominant benign traffic, maintaining high sensitivity without sacrificing the event-driven efficiency of the spiking network.

In this context, our main contributions are summarized as follows.

- A lightweight packet stream preprocessing pipeline is developed with a behavior-adaptive redundancy filter to compress highly repetitive traffic and group packets into pseudo sessions, with the PON classifier operating without payload or raw IP address access in the inference stage.
- A PON architecture using 1D convolution is constructed with the SAD module designed to mitigate temporal spike dilution, using a learned gating mechanism to amplify sporadic attack signatures and improve multiclass classification.

The remainder of this paper is organized as follows. Section 2 presents the proposed methodology. Section 3 describes the experimental setup. Section 4 presents and discusses the evaluation results. Section 5 concludes the paper.

2. Methodology

2.1. Packet Stream Preprocessing with Redundancy Filter

Conventional flow-based intrusion detection systems aggregate statistical features over all packets belonging to the same five-tuple comprising source IP address, destination IP address, source port, destination port, and protocol after the network flow terminates. This design introduces several major limitations. First, classification can only occur after the flow ends, leading to significant detection latency, especially for persistent connections in IoT environments. Second, statistical aggregation discards the temporal dynamics of attack behaviors, such as the rapid escalation rate of denial of service attacks or the interleaving of benign and attack packets within the same time interval. Third, the volume imbalance between benign and attack traffic in real world environments also impedes robust detection.

The proposed system operates as a continuous packet stream compressor rather than a conventional flow extractor. Each retained packet corresponds to a single input timestep for the PON model. Classification is performed after every window of T incoming packets without depending on the termination state of the connection or the flow identification of the packets.

An important design constraint is that the PON classifier is only allowed to access the packet headers during inference, specifically the first 64 bytes of the IP and transport-layer headers, without reading any payload content during the classification stage. This restriction is motivated by three practical considerations. First, the payload content of IoT traffic can be encrypted, making payload analysis infeasible without decryption keys. Second, DPI techniques that require payload access incur extremely high processing latency and computational overhead at the network-edge. Third, the 64-byte header limit yields a fixed size, compact feature vector suitable for the processing power and memory of resource-constrained edge devices in IoT environments. Section 2.2 describes in detail how these header bytes are organized into the input feature vector for the model.

CIC-IoT2023 collects benign activity and attack scenarios under controlled, scenario-specific experimental procedures. Processing these captures independently would expose the detector mainly to long sequences dominated by one traffic regime, whereas an operational IoT network contains malicious packets interleaved with legitimate device activity and background services. We therefore superimpose one recorded benign trace on each recorded attack trace. The benign trace serves as legitimate competing background traffic, wh ich re quires th e de tector to identify attack behavior at the packet and window levels rather than infer a label from an isolated capture.

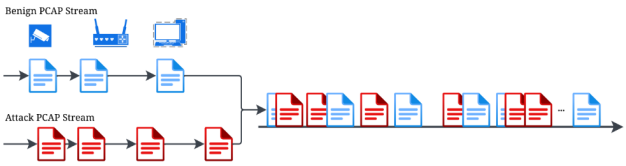

Figure 1. Mixed stream construction process from separate benign and attack sources

The process of constructing a mixed packet stream from a pair of files $(\mathcal{F}_B, \mathcal{F}_A)$ consists of three steps.

The first step is reading and normalizing timestamps. The system reads all packets from the benign file $\mathcal{F}_B$ and the attack file $\mathcal{F}_A$ to generate two annotated streams.

$$\mathcal{B} = \left\{(p,\ t - t^{B}_{\min},\ 0)\right\}_{p\in\mathcal{F}_B}$$
$$\mathcal{A} = \left\{(p,\ t - t^{A}_{\min} + \delta,\ c)\right\}_{p\in\mathcal{F}_A} \tag{1}$$

where $t^{B}_{\min}$ and $t^{A}_{\min}$ are the start times of the respective files, $\delta \geq 0$ is a time offset applied to adjust the starting time of the attack events relative to the benign traffic to create interleaving between the two streams, and $c \in \{1, \ldots, C\}$ is the corresponding attack class label.

For each source PCAP, timestamps are represented relative to the first packet in that file. The attack trace is then shifted by the offset in Eq. (1), and the two traces are merged in ascending relative-time order. This operation preserves the packet order and interarrival intervals within each original trace. It does not shuffle packets, resample traffic, scale packet timing, or modify packet contents. The offset therefore controls only the relative starting position of the two recorded traces.

The second step is merging and sorting. The two streams are merged and sorted in ascending chronological order to produce the mixed stream $\mathcal{M}$.

$$\mathcal{M} = \mathrm{sort}_t\left(\mathcal{B} \cup \mathcal{A}\right) \tag{2}$$

The third step is redundancy filtering. The filter RF(·) described in Algorithm 1 is applied to $\mathcal{M}$ to discard repetitive packets that do not exhibit state changes, yielding the compressed stream $\mathcal{P}'$:

$$\mathcal{P}' = \mathrm{RF}(\mathcal{M}), \qquad |\mathcal{P}'| \ll |\mathcal{M}| \tag{3}$$

Figure 1 illustrates the mixed stream construction process from two separate traffic sources.

The Redundancy Filter RF pursues two simultaneous objectives without using label information. The first

objective is to reduce data volume by discarding repetitive packets that do not carry state changes relative to the most recently retained packet of the same pseudo-session. The second objective is to increase the density of significant packets in the stream by avoiding the processing of highly repetitive packets, exploiting the characteristic that attack traffic typically produces higher state change dynamics than normal benign traffic.

To enable per connection comparison without relying on raw IP addresses, which prevents device specific overfitting, each packet p is mapped to a pseudo-session key.

$$key(p) = \big(\text{eth\_type}(p),\ \text{proto}(p),\ \min(\text{port\_src}(p),\ \text{port\_dst}(p))\big) \tag{4}$$

Equation 4 groups packets by network protocol, transport protocol, and approximate service port so that the redundancy filter can compare similar traffic using the same state s_k. The minimum source or destination port provides a direction-independent service identifier, while IP addresses are excluded to avoid device-specific grouping.

IP addresses are excluded from the pseudo-session key and therefore do not directly affect packet grouping in the redundancy filter. Service-port transformations modify the key itself, whereas NAT-like and source-IP transformations affect the separately computed source-IP diversity feature used by PON.

The system maintains a per-key state s_k recording the behavioral features of the most recently retained packet. Each incoming packet p with key k is compared against s_k using four state change criteria, represented by indicator functions $\mathbf{1}[\cdot]$ which equal one if the condition holds and zero otherwise. The notation $s_k.t^*$ denotes the field value from the previously retained packet, and $\Delta t = \text{ts}(p) - s_k.t^*$ is the elapsed time since the last retention. For packet p, $\text{ts}(p)$ denotes its arrival timestamp on the normalized mixed-stream clock. It represents time, not a packet index.

$$c_1 = \mathbf{1}[\Delta t \geq \Theta_{\text{idle}}] \tag{5}$$

$$c_2 = \mathbf{1}[\text{flags}(p) \neq s_k.f^*] \tag{6}$$

$$c_3 = \mathbf{1}\left[\frac{|\text{len}(p) - s_k.\ell^*|}{\max(s_k.\ell^*,\ 1)} \geq R_\ell\right] \tag{7}$$

$$c_4 = \mathbf{1}\Big[|H(p) - s_k.H^*| \geq E_\delta\Big] \tag{8}$$

where $H(p)$ is the Shannon entropy of the packet payload. When c_4 is enabled, the filter reads the payload bytes transiently to compute this scalar statistic. It does not parse application fields or inspect signatures. Raw payload bytes are discarded after the keep/drop decision and are not forwarded to PON. Here, Θ_{idle} is the idle gap threshold, R_ℓ is the length change ratio threshold, and E_δ is the entropy deviation threshold. Semantically, criterion c_1 captures temporal inactivity, c_2 detects protocol state transitions through TCP flags, while c_3 and c_4 reflect variation in size and payload statistical characteristics. The aggregate state change score evaluating the behavioral anomaly of the packet is computed as:

$$\sigma(p,\ s_k) = \sum_{i=1}^{4} c_i \tag{9}$$

The state change score σ is compared against the behavioral adaptive threshold (BAT) τ_{BAT}. To track the traffic dynamics of each pseudo-session key, the BAT mechanism maintains two exponential moving averages: the average time interval $\mu_{\Delta t}$ and the average behavior change score μ_σ. Throughout Eqs. (4)–(9), p denotes the packet currently processed by the filter. In Eqs. (10) and (11), the superscript (n) denotes the n-th update of the per-key moving-average state rather than a different packet variable. The update equations are defined as follows.

$$\mu_{\Delta t}^{(n)} = (1-\gamma)\cdot\mu_{\Delta t}^{(n-1)} + \gamma\cdot\Delta t \tag{10}$$

$$\mu_{\sigma}^{(n)} = (1-\gamma)\cdot\mu_{\sigma}^{(n-1)} + \gamma\cdot\sigma(p,\ s_k) \tag{11}$$

where the decay factor γ defaults to 0.1, and the time interval is the difference between arrival timestamps of consecutive packets of the same key. From these indices, the adaptive threshold τ_{BAT} is computed via a non-linear exponential relation:

$$\tau_{BAT} = \tau_{base} + e^{-\frac{\mu_{\Delta t}}{100.0}}\cdot\mu_\sigma \tag{12}$$

where τ_{base} is the base retention threshold defaulting to 1.0, and the adaptive threshold is clipped within $[\tau_{base}, 4.0]$. This mechanism increases the threshold during bursty traffic where the average time interval $\mu_{\Delta t}$ is small to aggressively compress repetitive packets, while dropping the threshold back to the base value during silent periods to capture sparse, sporadic packets. Figure 2 illustrates this per-key adaptive threshold filtering process under different traffic densities. The final retention decision is determined by:

$$\text{keep}(p) = \begin{cases} \top & \text{if } s_k.seen \leq K_{\text{pre}} \\ & \text{or } s_k.skip \geq K_{\text{skip}} \\ \mathbf{1}[\sigma(p,\ s_k) \geq \tau_{BAT}] & \text{otherwise} \end{cases} \tag{13}$$

where $s_k.seen$ is the packet count for key k and $s_k.skip$ is the consecutive skipped packet count. The first condition guarantees that initial packets are retained to establish the reference state, while the second condition forces periodic updates when traffic is fully stationary. Algorithm 1 details the complete filtering procedure.

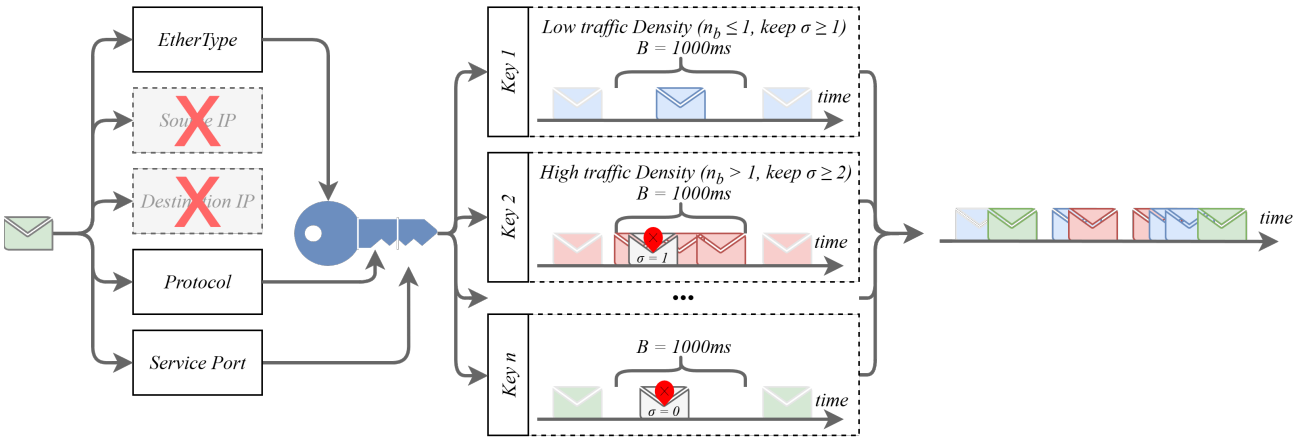

Figure 2. Per-Key adaptive threshold filtering process with independent timelines

The compressed packet stream of each traffic flow is divided into non overlapping, fixed size windows of T consecutive packets in temporal order. Each window serves as an independent training sample processed by the PON model. In the operational phase, the system classifies every window of T incoming packets without requiring connection identity or flow state.

Since each mixed stream is constructed by combining exactly one attack PCAP with one benign PCAP, each stream contains at most one attack class c interleaved with benign packets. A window, therefore, consists of either purely benign traffic or a mixture of class c and benign traffic. The labeling criterion arises from a conservative intrusion detection principle: the presence of any single attack packet is sufficient to classify the entire window as an attack, because a real intrusion does not need to constitute a majority of observed packets to cause harm. Formally, the label of window i is assigned as follows.

$$y_i = \begin{cases} 0 & \text{if } \hat{y}_{i,t} = 0 \ \forall\, t \in \{1, \ldots, T\} \\ c & \text{if } \exists\, t : \hat{y}_{i,t} = c \neq 0 \end{cases} \tag{14}$$

where $\hat{y}_{i,t}$ is the ground truth label of the packet at position t in window i.

Each window is additionally assigned a confidence weight w_i measuring the fraction of attack packets within the window.

$$w_i = \frac{1}{T} \sum_{t=1}^{T} \mathbf{1}[\hat{y}_{i,t} \neq 0] \tag{15}$$

A window containing only attack packets has $w_i = 1$, a fully benign window has $w_i = 0$, and mixed windows receive intermediate values. The weight w_i is used in the dual level loss function described in Subsection 2.3 to adjust the contribution of each window to the loss function, helping the model focus on samples with clear behavior patterns.

The dataset is partitioned at the stream level to ensure independent evaluation. Each pair of benign and attack files is assigned in its entirety to the training, test, or evaluation partition, so that no packets from the same recording session appear in more than one split. A stratified allocation based on per class stream counts yields an 80/10/10 stream level split while guaranteeing that every attack class, including rare categories with as few as four streams, appears in all three partitions.

2.2. Lightweight Header Feature Engineering

Each packet retained by the Redundancy Filter is represented by a 65-dimensional feature vector $\mathbf{x} \in \mathbb{R}^{65}$ constructed without accessing any payload bytes, thereby avoiding the high memory footprint of DPI. The header-only loader uses the normalized 64-byte header and the causal source-IP diversity ratio. It ignores payload-derived statistics that may be present in the intermediate extraction file. The two feature groups are described below.

The first 64 dimensions are obtained by concatenating three components. The first component is the EtherType field occupying 2 bytes that identifies the network layer protocol. The second component is the address stripped IP header, specifically the first 12 bytes of the IP header comprising the version, IHL, DSCP, total length, identification, IP flags, TTL, protocol number, and IP checksum, followed by any IP options bytes up to the end of the IP header. Source and destination IP addresses are deliberately excluded for three reasons. The first reason is to ensure the model focuses purely on traffic behaviors rather than hardware identifiers. The second reason is to prevent device specific overfitting, because a model that memorises IP addresses associated with attack traffic in the training set will fail to generalise to unseen addresses in deployment. The third reason is to prevent data leakage arising

Algorithm 1 Redundancy Filter Algorithm

Input: P — mixed stream sorted by timestamp
Parameters: Θ_{idle}, R_ℓ, E_δ representing idle gap, length ratio, and entropy thresholds
τ_{base} — base threshold, γ — decay factor for EMA
K_{pre} — mandatory initial keep count, K_{skip} — max consecutive skip
Output: $P' \subseteq P$ — filtered stream
$\mathcal{S} \leftarrow \emptyset$ ▷ per-key state table indexed by key
for each $p \in P$ **do**
  $k \leftarrow (\text{eth\_type}(p),\ \text{proto}(p),$
    $\min(\text{port\_src}(p),\ \text{port\_dst}(p)))$
  if $k \notin \mathcal{S}$ **then** ▷ new pseudo-session key, initialize state
    initialize $\mathcal{S}[k]$: $seen$=0, $skip$=0, all cached fields = $\perp$, $\mu_{\Delta t}$=0.0, μ_σ=0.0
  end if
  $s \leftarrow \mathcal{S}[k]$; $s.seen \leftarrow s.seen + 1$
  if $s.t^* \neq \perp$ **then** $\Delta t \leftarrow \text{ts}(p) - s.t^*$ **else** $\Delta t \leftarrow 0.0$ **end if**
  if $s.seen \leq K_{\text{pre}}$ **then** $keep \leftarrow \top$
  else if $s.skip \geq K_{\text{skip}}$ **then** $keep \leftarrow \top$
  else
    $\sigma \leftarrow 0$
    if $s.t^* \neq \perp$ and $\Delta t \geq \Theta_{\text{idle}}$ **then** $\sigma \leftarrow \sigma + 1$ **end if**
    if $s.f^* \neq \perp$ and $\text{flags}(p) \neq s.f^*$ **then** $\sigma \leftarrow \sigma + 1$ **end if**
    if $s.\ell^* \neq \perp$ and $\frac{|\text{len}(p) - s.\ell^*|}{\max(s.\ell^*, 1)} \geq R_\ell$ **then** $\sigma \leftarrow \sigma + 1$ **end if**
    if $s.H^* \neq \perp$ and $|H(p) - s.H^*| \geq E_\delta$ **then** $\sigma \leftarrow \sigma + 1$ **end if**
    $s.\mu_{\Delta t} \leftarrow (1-\gamma) \cdot s.\mu_{\Delta t} + \gamma \cdot \Delta t$
    $s.\mu_\sigma \leftarrow (1-\gamma) \cdot s.\mu_\sigma + \gamma \cdot \sigma$
    $\tau \leftarrow \tau_{base} + e^{-s.\mu_{\Delta t}/100.0} \cdot s.\mu_\sigma$
    $\tau \leftarrow \max(\tau_{base},\ \min(4.0,\ \tau))$
    $keep \leftarrow (\sigma \geq \tau)$
  end if
  if $keep$ **then**
    $P' \leftarrow P' \cup \{p\}$
    $s.t^* \leftarrow \text{ts}(p)$
    $s.\ell^* \leftarrow \text{len}(p)$
    $s.f^* \leftarrow \text{flags}(p)$
    $s.H^* \leftarrow H(p)$
    $s.skip \leftarrow 0$
  else
    $s.skip \leftarrow s.skip + 1$
  end if
end for
return P'

from the dataset collection structure of CIC-IoT2023, in which each attack scenario is recorded from a fixed set of hosts, causing IP addresses to become scenario identifiers rather than behavioural attack signals. The third component is the port stripped transport header, comprising TCP or UDP fields starting from byte offset four onward and including sequence number, acknowledgement number, TCP window size, and TCP flags, while source and destination ports are excluded for the same overfitting reason, given that fixed port numbers associated with a specific attack scenario would not generalise to real world traffic. The full concatenated sequence is padded or truncated to exactly 64 bytes and normalised to $[0, 1]$ by dividing each byte by 255.

A causal rolling window of $K = 32$ consecutive packets is maintained to compute the fraction of distinct source IP addresses among the K most recent arrivals.

$$x_{\text{src}}^{(t)} = \frac{\left|\left\{\text{src\_ip}_j : j \in [\max(0, t-K+1),\ t]\right\}\right|}{K} \tag{16}$$

This scalar is appended as the 65th dimension. A Denial of Service attack originating from a single host produces a ratio near $1/K$, whereas a Distributed Denial of Service campaign involving multiple spoofed sources or botnet nodes drives the ratio toward 1.0. The computation is strictly causal because only the current packet and past packets within the rolling window are used.

2.3. PON Architecture and Training Methodology

The model processes a window of T consecutive packet features as a time sequence to classify both the window label and the label at each individual timestep.

Core Model Architecture. The proposed PON utilizes a multi-layer structure with a 1D convolution block to perform real-time processing. The core architecture consists of three components. a linear layer, a convolution block, and spiking LIF neuron layers.

The first component is a linear layer that maps the 65-dimensional input feature vector to a 32-dimensional hidden space. This layer remains at full float32 precision during training and inference. Quantizing this initial mapping layer reduces the detection accuracy of rare attack classes with sparse samples. The remaining layers in the PON model, including the convolution layer and the subsequent spiking linear layers, are quantized to ternary values using TTQ [25]. This process quantizes real valued weights to three learnable states, which are negative W_n, zero, and positive W_p, with the quantization threshold set to five percent of the maximum weight magnitude of each layer, reducing model size and enabling faster computation on edge hardware.

The second component is a convolution block with a kernel size $k = 3$ and a dilation factor $d = 1$ along

the temporal axis T, using symmetric padding on both sides. A Rectified Linear Unit, ReLU, activation is applied after the convolution layer. This convolution layer has a receptive field of three timesteps, while longer temporal context is accumulated by the subsequent LIF neuron layers through membrane potential integration.

The third component is the spiking LIF neuron layers. Two linear layers combined with spiking neurons are placed after the convolution block: a layer mapping from 32 to 32 dimensions, and another mapping from 32 to 16 dimensions. These layers utilize standard LIF neurons [26, 27] with a decay factor $\tau = 0.9$ and a firing threshold $\theta = 1.0$. During inference, the LIF neuron compares its membrane potential with the threshold to generate a binary spike, which is either zero or one. This operation replaces expensive matrix multiplications with sparse accumulations. The model does not use float valued skip connections between these spiking layers to preserve the temporal sparsity of binary spikes. For weight updates during training, since the binary spiking activation function has a zero gradient almost everywhere, a surrogate gradient method [28] using an arc tangent function is employed to approximate the derivative and enable error backpropagation.

SAD Module. When the ratio of attack packets within a window is small, the membrane potential of the LIF neurons accumulates spikes from both benign and attack traffic, leading to temporal dilution. This dilution degrades the classification accuracy of the readout layer because the conventional temporal average operation degrades the signal of sparse attack packets. The SAD module is introduced to mitigate this effect by routing information through two parallel paths. an amplification pathway and a sparse attention pathway.

Let $\mathbf{h}[t] \in \mathbb{R}^D$ denote the output state of the second spiking layer at timestep t, where the hidden dimension D is equal to 16.

The first path is the amplification pathway, which detects and enhances anomalous timesteps. This path computes a depthwise sliding mean pool along the temporal dimension with a local window size $K_{\text{sad}} = 8$.

$$\bar{\mathbf{h}}[t] = \frac{1}{\min(t,\, K_{\text{sad}})} \sum_{k=0}^{\min(t,\, K_{\text{sad}})-1} \mathbf{h}[t-k] \tag{17}$$

Equation (17) computes a moving average state $\bar{\mathbf{h}}[t]$ representing the local context of the packet stream prior to timestep t. The min function ensures safe calculation during the initial steps where the accumulated packet count is smaller than eight.

The absolute deviation from this local average represents sudden spike density variations and is used to compute an anomaly score.

$$\text{score}[t] = \sigma\left(\mathbf{w}_s^\top \left|\mathbf{h}[t] - \bar{\mathbf{h}}[t]\right|\right) \tag{18}$$

where $\mathbf{w}_s \in \mathbb{R}^D$ is a learnable weight vector and σ represents the sigmoid activation function. The absolute difference measures the behavior anomaly at timestep t, which is projected and normalized by the sigmoid function to an anomaly score between 0.0 and 1.0.

This anomaly score scale is used to scale the initial neuron state via a multiplicative gate.

$$\tilde{\mathbf{h}}[t] = \mathbf{h}[t] \cdot (1 + \alpha \cdot \text{score}[t]) \tag{19}$$

where α is a learnable scaling parameter initialized to zero. For normal timesteps with low scores, the original state is preserved. For anomalous timesteps with high scores, the gating operation amplifies the state magnitude to enhance the visibility of sparse attack features relative to surrounding benign packets.

The second path is the sparse attention pathway, which isolates attack signals and filters out noise. This path computes a sparse attention weight distribution $\mathbf{a}$ using a softmax function with a scaling temperature factor of ten.

$$\mathbf{a} = \text{softmax}(10 \cdot \text{score}) \tag{20}$$

The factor of ten sharpens relative differences among the timestep scores before normalization. Since the softmax transformation is monotonic, timesteps with larger scores receive greater relative attention within the same window. This behavior is evaluated directly in Section 4.4. The output of this pathway is the weighted sum of the states projected to the classification space.

$$\mathbf{z}_{\text{sad}} = \mathbf{W}_r \sum_{t=1}^{T} a[t]\,\mathbf{h}[t] \tag{21}$$

where $\mathbf{W}_r$ is the weight matrix of the attention pathway.

The final classification decision $\mathbf{z}$ is obtained by a linear summation of the outputs from both pathways.

$$\mathbf{z} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{W}_o\, \tilde{\mathbf{h}}[t] + \mathbf{z}_{\text{sad}} \tag{22}$$

where $\mathbf{W}_o$ is the primary classification weight matrix. This summation combines the global context from the first pathway with the sparse anomalous signal from the second pathway, optimizing the detection of sparse attacks hidden within long observation windows. The SAD module adds approximately 733 learnable parameters and operates on temporal windows to support online deployment.

3. Experimental Setup

3.1. Dataset CIC-IoT2023

The experiments in this study are conducted on the CIC-IoT2023 dataset published by Neto *et al.* [3], which represents a realistic IoT environment consisting of 105 IoT devices. The dataset captures 33 attack scenarios categorized into DDoS, DoS, reconnaissance, web attacks, brute force, spoofing, and Mirai, recorded as raw PCAP files. In our proposed framework, packet streams are extracted directly from the raw PCAP files and grouped using the pseudo-session keys described in Section 2.1. The preprocessing pipeline processes a total of 1,220 streams containing 478,279,838 raw packets. After passing through the behavior-adaptive redundancy filter, the retained packets, totaling 96,756,769 packets, are partitioned into 755,308 data windows. This represents a traffic reduction of 79.77 percent, achieved by skipping 381,523,069 redundant packets that did not exhibit significant behavioral state changes with a window size $T = 256$ and a stride $S = 128$. The dataset is split into 80% for training, 10% for validation, and 10% for testing at the stream level to prevent temporal data leakage.

Table 1 presents the distribution of windows across the eight primary categories in the preprocessed dataset.

Table 1. Distribution of data windows across the eight primary categories of the preprocessed CIC-IoT2023 dataset

Category	Total Windows	Percentage [%]
Benign	548,114	72.57
DDoS	108,999	14.43
DoS	46,762	6.19
Recon	16,915	2.24
Web	6,256	0.83
BruteForce	1,706	0.23
Spoofing	4,021	0.53
Mirai	22,535	2.98
Total	**755,308**	**100.00**

Due to the nature of the redundancy filter and the mixed packet stream structure, the ratio of attack packets within each window exhibits significant variation and sparsity across different attack categories. Table 2 compiles the statistical indicators including the mean, minimum, maximum, and standard deviation of the attack packet ratios for each attack category.

Figure 3 illustrates the smooth distribution of data windows according to the attack packet ratio for each category. For DDoS and DoS attacks, the sample count concentrated strictly around 100%, demonstrating continuous attack transmission behaviors. Conversely, Web attacks exhibit a wider distribution across lower ratio ranges, demonstrating that attack packets appear sporadically and interleave densely with benign traffic within the same window.

Table 2. Statistical indicators of attack packet ratios across sparse attack windows

Category	Mean [%]	Min [%]	Max [%]	Std [%]
DDoS	90.97	0.39	100.00	14.96
DoS	97.67	0.78	100.00	7.27
Recon	81.92	0.39	100.00	23.74
Web	41.15	0.39	100.00	20.68
BruteForce	93.39	0.39	100.00	18.37
Spoofing	76.06	3.52	100.00	24.23
Mirai	74.80	0.39	100.00	15.50

To evaluate the intrusion detection capability comprehensively, the models are trained and evaluated under two distinct classification tasks. The first task is binary classification, where all seven attack categories are grouped into a single malicious class along with the benign class, suitable for resource-constrained edge deployments requiring rapid alarm triggering. The binary dataset consists of 548,114 benign windows, representing 72.57 percent, and 207,194 attack windows, representing 27.43 percent. The second task is multi-class classification, where the data windows are classified into all eight individual categories, providing detailed threat intelligence to support fine-grained response strategies in IoT systems.

3.2. Evaluation Metrics

To evaluate the classification performance at the window level on the test set, several standard evaluation metrics are employed. Due to the high class imbalance in the CIC-IoT2023 dataset, the Macro F1 score is designated as the primary evaluation metric.

The accuracy, Acc, is defined as

$$\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN} \tag{23}$$

The precision, also known as the positive predictive value, PPV, is defined as follows.

$$\text{PPV} = \frac{TP}{TP + FP} \tag{24}$$

The detection rate, representing the true-positive rate, TPR, is defined as follows.

$$\text{TPR} = \frac{TP}{TP + FN} \tag{25}$$

The false-positive rate, FPR, is defined as follows.

$$\text{FPR} = \frac{FP}{FP + TN} \tag{26}$$

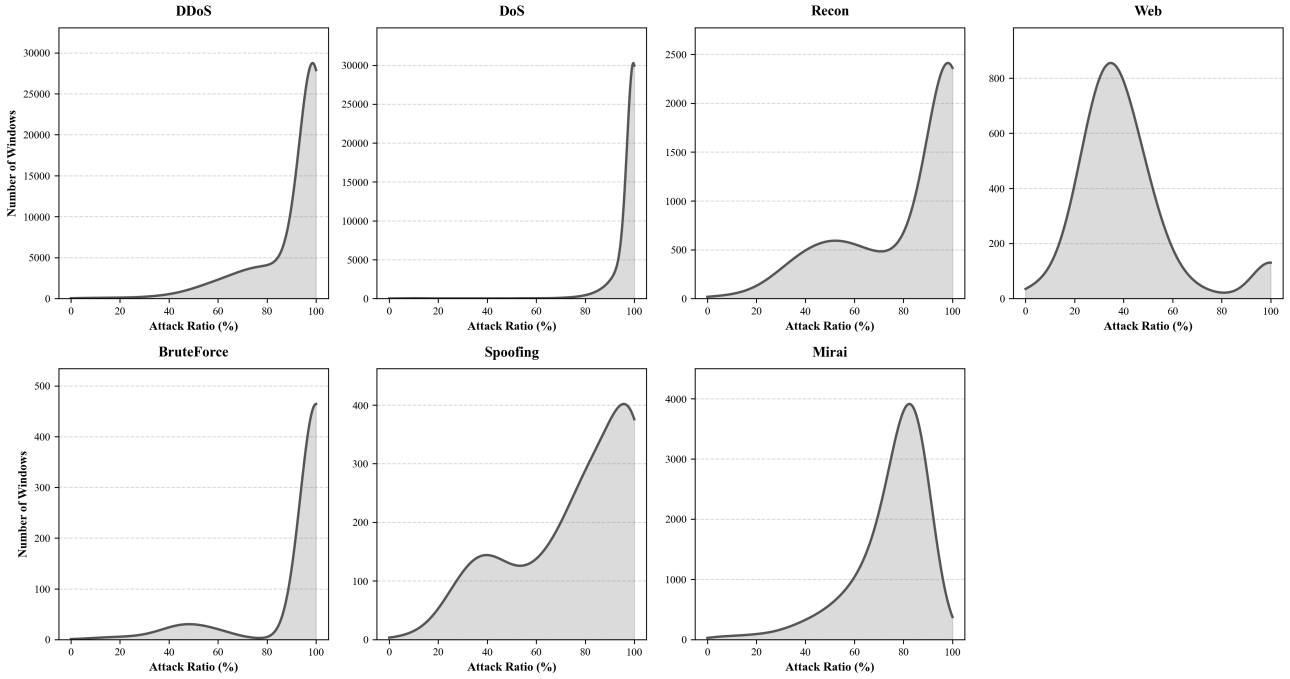

Figure 3. Distribution of data windows according to the ratio of attack packets for each attack category

The false-negative rate, FNR, is defined as follows.

$$\text{FNR} = \frac{FN}{TP + FN} \tag{27}$$

The Macro F1 score is computed by averaging the F1 scores across all individual classes.

$$\text{Macro F1} = \frac{1}{C} \sum_{c=1}^{C} 2 \cdot \frac{\text{PPV}_c \cdot \text{TPR}_c}{\text{PPV}_c + \text{TPR}_c} \tag{28}$$

where C is the number of classes, and TP, TN, FP, and FN represent the number of true-positive, true-negative, false-positive, and false-negative predictions, respectively.

3.3. Experimental Environment and Hyperparameters

All experiments are conducted on a high performance workstation equipped with an Intel Core i9-12900K processor, 96 GB of RAM, and an NVIDIA GeForce RTX 3090 GPU. The models are implemented in Python 3.10 using the PyTorch framework.

During the experiments, the PON model operates with a window size $T = 256$, and a 65-dimensional input feature vector representing the 64-byte normalized header and the 1-dimensional causal source IP ratio. The LIF neurons use a decay factor $\tau = 0.9$ as the membrane time constant. The model training utilizes a batch size of 512 and an initial learning rate of 10^{-3} optimized using the AdamW algorithm during the initial float32 training phase.

3.4. Baseline Models

To comprehensively evaluate the performance of our proposed model, we compare it against classic recurrent architectures and perform systematic ablation studies to quantify the contributions of individual structural components.

The recurrent reference baselines include a Recurrent Neural Network (RNN) the LSTM network, and a Gated Recurrent Unit (GRU) network. To ensure a fair comparison regarding model capacities and parameter counts, each of these recurrent models is configured with two recurrent layers and a hidden dimension equivalent to that of the proposed PON model.

Furthermore, systematic ablation studies are established based on the PON framework. The proposed TTQ PON model represents the complete architecture optimized with ternary weights. The proposed float32 PON model represents the full architecture trained entirely at single-precision float32 without quantization to analyze the exact performance impact of the TTQ mechanism. Finally, the proposed PON model without SAD completely removes the SAD module to verify its empirical impact on rare attack classification where temporal dilution is hypothesized to occur.

4. Results and Discussion

This section presents the evaluation of the proposed PON system across four aspects, which are binary anomaly detection, multi-class attack classification with per class analysis, ablation studies of the SAD

Table 3. Binary classification performance comparison on the mixed packet streams extracted from CIC-IoT2023 PCAP files

Model	Acc	MacroF1	PPV	TPR	FPR	FNR
CNN+LSTM [11]*	99.56%	99.56%	99.57%	99.56%	–	–
Transformer [11]*	99.52%	–	99.54%	99.52%	–	–
CNN [10]*	99.40%	99.41%	99.43%	99.40%	–	–
RNN baseline	97.96%	97.52%	100.00%	93.21%	0.00%	6.79%
LSTM baseline	99.14%	98.97%	100.00%	97.15%	0.00%	2.85%
GRU baseline	99.24%	99.09%	100.00%	97.47%	0.00%	2.53%
PON TTQ + SAD	**98.55%**	**98.26%**	**100.00%**	**95.19%**	**0.00%**	**4.81%**

Table 4. Overall eight-class detection performance comparison on the mixed packet streams extracted from CIC-IoT2023 PCAP files

Model	Acc	MacroF1	PPV	TPR	FPR	FNR
CNN+RNN [11]*	99.38%	70.50%	99.63%	99.43%	32.58%	0.57%
CNN+LSTM [11]*	99.67%	76.26%	99.84%	99.72%	13.97%	0.28%
Transformer [11]*	99.68%	78.18%	99.82%	99.73%	15.86%	0.27%
DNN [10]*	99.02%	69.14%	99.59%	99.11%	17.17%	0.89%
LSTM [10]*	85.98%	58.31%	99.93%	98.95%	2.92%	1.05%
CNN [10]*	99.10%	70.96%	99.62%	99.19%	16.20%	0.81%
RNN baseline	94.57%	84.94%	91.10%	81.91%	3.44%	18.09%
LSTM baseline	97.21%	92.67%	95.27%	90.71%	1.93%	9.29%
GRU baseline	97.37%	93.53%	94.99%	91.26%	2.07%	8.74%
PON float32	95.70%	88.55%	93.81%	85.70%	2.43%	14.30%
PON float32 + SAD	97.41%	93.01%	95.01%	91.38%	2.06%	8.62%
PON TTQ	95.37%	87.32%	92.64%	84.58%	2.89%	15.42%
PON TTQ + SAD	**97.06%**	**92.48%**	**94.54%**	**90.23%**	**2.24%**	**9.77%**

module and TTQ quantization, and model resource efficiency.

4.1. Binary Classification Results

The binary classification task merges all seven attack categories into a single malicious class along with the benign class, providing a deployment-ready configuration for resource-constrained edge devices that require rapid alarm triggering without fine-grained attack identification. Table 3 compares the binary detection performance of the proposed quantized PON against the reference models and recurrent baselines on the mixed packet streams extracted from CIC-IoT2023 PCAP files at the window level. The reference models from the literature, marked with an asterisk, report results evaluated at the raw packet-level in their original publications, whereas all recurrent baselines and the proposed SNN models in our experiments are evaluated at the temporal window level.

The proposed PON TTQ + SAD model achieves a window accuracy of 98.55% and a Macro F1 score of 0.9826. The model achieves a perfect positive predictive value of 100.00% and an optimal false-positive rate of 0.00% at the window level, meaning that the model produces zero false alarms on benign traffic windows while maintaining a high attack detection rate of 95.19%. This result exceeds the RNN baseline at 97.96% accuracy and 0.9752 Macro F1, while remaining slightly below the LSTM at 99.14% and the GRU at 99.24%. The small gap of 0.59 to 0.69 percentage points in window accuracy compared with the two stronger recurrent baselines represents a deliberate and worthwhile trade-off. All models are trained and evaluated on the same compressed packet streams to ensure a completely fair comparison. Under this identical setting, the proposed PON model achieves competitive classification accuracy while offering substantial savings in computational overhead. Unlike the recurrent baselines that rely on continuous value floating point matrix multiplications at every timestep, the event-driven computational paradigm of the SNN replaces these expensive operations with sparse, conditional additions over quantized ternary weights, making it highly suitable for resource-constrained IoT edge devices.

4.2. Multiclass Detection Performance

Beyond binary anomaly detection, identifying the specific attack category is essential for enabling automated response strategies at the network-edge. Table 4 compares the overall eight-class detection performance of the proposed quantized PON model against recurrent baselines and published reference models on the mixed packet streams extracted from CIC-IoT2023 PCAP files. All four italicized configurations at the bottom of Table 4 represent different architectural variants of the proposed PON family, spanning single-precision float32 formats, TTQ versions, and sparse attention gate configurations. All experimental results from this study are reported at the window level to ensure consistent evaluation. The reference models from Tseng *et al.* [11] and Becerra *et al.* [10] are included for relative comparison only, as they report results at the raw packet-level on their own preprocessed datasets without the proposed behavior-adaptive redundancy filter.

When compared against the recurrent baselines that are trained and evaluated under identical conditions at the window level, the proposed PON TTQ + SAD model achieves competitive performance. The window accuracy reaches 97.06%, closely matching the GRU at 97.37% and the LSTM at 97.21%, while surpassing the RNN at 94.57%. The Macro F1 score of the proposed model reaches 92.48 percent, approaching the GRU at 93.53 percent and the LSTM at 92.67 percent, while substantially exceeding the RNN at 84.94 percent. The attack detection rate expressed as TPR reaches 90.23%, comparable to the GRU at 91.26% and the LSTM at 90.71%, while far exceeding the RNN at 81.91%. The false-positive rate of the proposed model achieves 2.24%, close to the LSTM at 1.93% and the GRU at 2.07%.

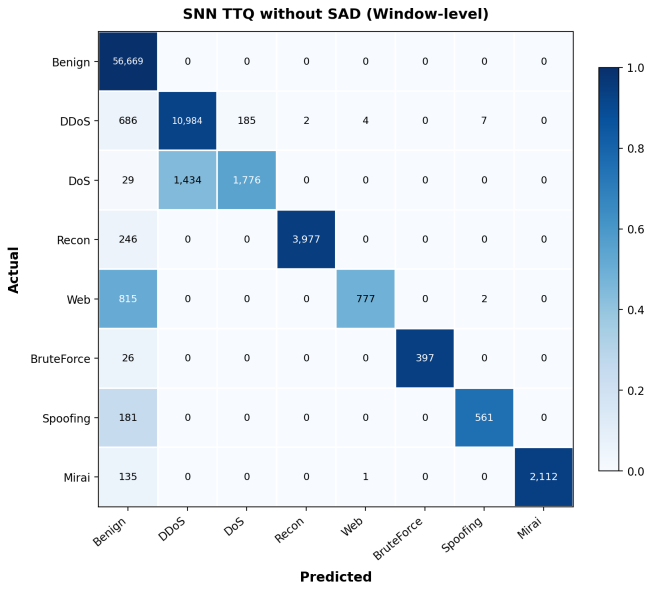

Figure 4. Confusion matrix of the PON TTQ model without SAD at the window level

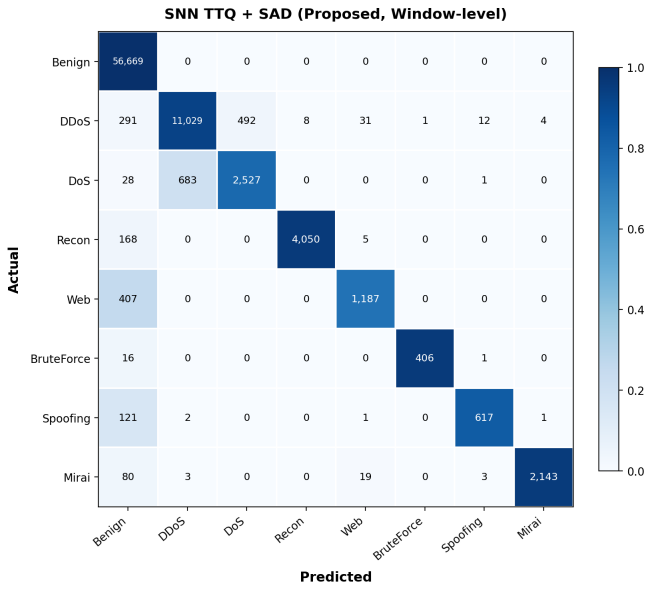

Figure 5. Confusion matrix of the proposed PON TTQ + SAD model at the window level

When compared with the reference models from recent publications, the Macro F1 score of the proposed PON TTQ + SAD at 92.48 percent substantially exceeds the Transformer model of Tseng *et al.* [11] at 78.18 percent and the CNN model of Becerra *et al.* [10] at 70.96 percent. Although the reference models report very high overall accuracy values ranging from 99.10% to 99.68%, these metrics are dominated by the majority classes such as DDoS and DoS in the severely imbalanced dataset. Analysis of the confusion matrices reported in those studies reveals that their detection capability for rare attack classes is critically poor. The CNN model of Becerra *et al.* completely fails to detect Web attacks with zero correct predictions and only detects 17.39% of BruteForce attacks. The Transformer model of Tseng *et al.* detects only 29.65% of Web attacks and 17.74% of BruteForce attacks.

To provide a detailed per class analysis, Figure 4 and Figure 5 present the confusion matrices of the PON TTQ model without and with the SAD module at the window level. Each cell shows the absolute count, and the color intensity represents the row normalized proportion.

The confusion matrix in Figure 5 demonstrates that the proposed model achieves high correct classification rates across most classes. The primary source of confusion occurs between the two denial of service classes, DoS and DDoS. Specifically, 683 DoS windows are misclassified as DDoS and 492 DDoS windows are misclassified as DoS. This confusion is expected because these two attack types share similar header level patterns and differ primarily in the degree of source address diversity, which is captured only indirectly through the causal source IP ratio feature. The Web attack class also records 407 windows misclassified as Benign because connection establishment packets from web based attacks share header characteristics with normal web browsing traffic. The proposed model achieves competitive detection rates for rare attack classes that are typically missed by conventional approaches. The Web attack detection rate reaches 74.47% at the window level, the BruteForce detection rate reaches 95.75%, the Recon class achieves 95.90%, and the Spoofing class achieves 83.04%.

4.3. Robustness under pseudo-session-key perturbations

To evaluate the robustness of the pseudo-session key, we reconstructed six controlled traffic configurations on the same 130-stream test partition used for the reported evaluation. The trained Binary and Group8 model weights used to produce the main results were kept fixed, and no model retraining was performed. The rebuilt baseline was first verified against direct evaluation of the original HDF5 dataset. The six configurations and their results are summarized in Table 5.

Binary Window Macro F1 remains between 92.83% and 96.96% across the perturbations, compared with 98.26% for the baseline. Low source ports and UDP/443 tunneling retain Binary Window Macro F1 values of 96.96% and 96.09%, respectively. Randomizing service ports assigns packets from the same service to different pseudo-session keys and reduces the packet-reduction

Table 5. Controlled robustness evaluation of the pseudo-session key and source-IP diversity feature using the fixed model weights from the main experiments

Mode	Retained packets	Reduction	Binary Win. F1	Binary TPR	Binary FPR	Group8 Win. F1	Group8 TPR	Group8 FPR
Baseline	10,376,946	79.49%	98.26%	95.19%	0.00%	92.48%	95.43%	0.00%
Low source port	7,342,332	85.49%	96.96%	90.97%	0.00%	88.87%	90.57%	0.33%
Random service port	47,594,420	5.93%	96.62%	95.71%	2.49%	68.49%	92.75%	0.35%
NAT collapse	10,376,946	79.49%	95.45%	87.73%	0.00%	64.19%	78.37%	0.00%
Source IP randomized	10,376,946	79.49%	92.83%	81.15%	0.02%	51.82%	97.76%	98.97%
Tunnel UDP/443	9,996,796	80.24%	96.09%	88.50%	0.02%	83.86%	82.39%	0.00%

rate from 79.49% to 5.93%. NAT collapse and source-IP randomization leave the retained-packet count unchanged because IP addresses are excluded from the pseudo-session key.

Under source-IP randomization, independently generated addresses drive the 32-packet source-IP diversity ratio toward 100% for both benign and attack traffic. Benign windows consequently resemble the distributed-attack patterns learned by the Group8 model, increasing Attack FPR to 98.97% and reducing Window Macro F1 to 51.82%. This deliberately extreme transformation identifies the operating boundary of the diversity feature rather than representing ordinary NAT traffic. Binary Window Macro F1 remains 92.83%, indicating that binary detection continues to use other header and temporal features. Payload encryption does not directly affect PON because the classifier does not read payload data. The UDP/443 configuration measures the effect of tunneling when a common outer header conceals the inner protocol and service port.

4.4. Ablation Study

The four PON variants in Table 4 form a complete two-factor ablation covering both the SAD module and the TTQ quantization mechanism, enabling independent evaluation of each component and their interaction.

The contribution of the SAD module is evaluated by comparing models with and without SAD under both quantization settings. Under TTQ quantization, removing SAD causes the Macro F1 to drop from 92.48 percent to 87.32 percent, a reduction of 5.16 percentage points. The attack TPR decreases from 90.23% to 84.58% and the FNR increases from 9.77% to 15.42%. Under float32 precision, the float32 model without SAD achieves a Macro F1 of 88.55 percent, while the float32 model with SAD reaches 93.01 percent. This result indicates that the SAD module provides a substantial performance benefit under both full precision and quantization settings by successfully amplifying sparse attack features that would otherwise be suppressed. Comparing Figure 4 with Figure 5 further reveals the impact on rare-class detection at the window level. The Web attack class records a decrease in correctly classified windows from 1,187 down to 777, corresponding to an increase in the miss rate from 25.53% to 51.25%. The DoS class records a decrease in correct predictions from 2,527 to 1,776 due to massive misclassification into the DDoS class, with 1,434 DoS windows misclassified as DDoS compared to 683 windows in the full model.

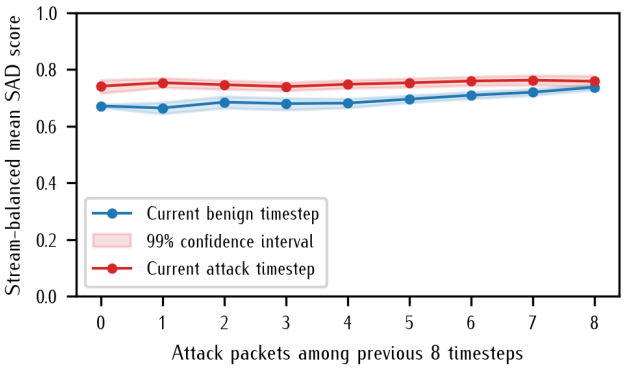

Figure 6. Context-conditioned SAD scores by the number of attack packets among the preceding eight timesteps. Shaded regions denote 99% confidence intervals

To evaluate the anomaly score in Equation 18 independently, we analyzed 10,375,906 unique packet timesteps from 130 test streams. Each physical timestep was counted once despite the overlap between adjacent sliding windows. For each timestep, the local context was defined by the number of attack packets among the preceding eight positions. Packet labels were used only after inference for this analysis and were not inputs to SAD. At every context level, the mean score of the current attack timestep was higher than that of the current benign timestep. The mean difference ranged from 0.0207 to 0.0775, and all corresponding 99% confidence intervals remained above zero. The separation was largest in sparse and mixed contexts, where attack information is more susceptible to temporal dilution, and became smaller when attack traffic already dominated the local context. Since Equation 20 monotonically maps the scores to attention weights, these results show that SAD assigns greater relative attention to attack timesteps under the same local temporal context. The corresponding context-conditioned scores are shown in Figure 6.

The contribution of the TTQ mechanism is evaluated by comparing ternary models against their float32 counterparts under both SAD configurations. When SAD is enabled, the TTQ model achieves a Macro F1 of 92.48 percent compared with 93.01 percent for its float32 counterpart, showing a negligible quantization loss of only 0.53 percentage points, while the window accuracy remains comparable at 97.06% compared with 97.41% for the float32 counterpart. The attack TPR is 91.38% for the float32 model and 90.23% for the ternary model, while the FPR is comparable at 2.06% for the float32 model and 2.24% for the ternary model. When SAD is disabled, the TTQ model achieves comparable performance to the float32 model with a Macro F1 of 87.32 percent versus 88.55 percent. The TTQ mechanism quantizes the real valued weights of the convolution layer and the two spiking linear layers to three learnable states: negative W_n, zero, and positive W_p, with the quantization threshold set to five percent of the maximum weight magnitude of each layer. The initial input mapping layer remains at float32 precision because quantizing this layer degrades the detection of rare attack classes with sparse training samples. These results demonstrate that TTQ quantization provides its strongest benefit in combination with SAD, where the ternary regularization and the SAD gating mechanism create a synergistic effect that substantially exceeds either component alone.

4.5. Model Resource Efficiency

The parameter count of each layer in the proposed SNN is derived from the architectural dimensions introduced in Section 2.3. The input dimension is 65, the hidden state dimension is $H = 32$, the spiking output dimension is $D = 16$, the convolution kernel size is $k = 3$, the SAD local window size is $K_{\text{sad}} = 8$, and the number of output classes is $C = 8$.

$$N_{\text{input}} = 65 \cdot H + 3H = 2176 \quad (29)$$

$$N_{\text{conv}} = H^2 k + 2H + 2 = 3138 \quad (30)$$

$$N_{\text{sad}} = 65(K_{\text{sad}} + 1) + CD + C + 2 = 723 \quad (31)$$

$$N_{\text{spiking}} = H(H + D) + 3(H + D) + 4 = 1684 \quad (32)$$

$$N_{\text{readout}} = CD + C = 136 \quad (33)$$

where the two additional constants in N_{conv} account for the positive and negative TTQ scaling factors, and the four constants in N_{spiking} account for the scaling factors of the two spiking linear layers. Summing across all layers gives the total parameter count.

$$\begin{aligned} N_{\text{total}} &= N_{\text{input}} + N_{\text{conv}} + N_{\text{sad}} + N_{\text{spiking}} + N_{\text{readout}} \\ &= 2176 + 3138 + 723 + 1684 + 136 = 7857 \end{aligned} \quad (34)$$

Table 6. Cortex-A53 inference: Float32 versus ternary TTQ PON

Metric	Float32	Ternary TTQ	Change
Latency (ms/window)	17.501	12.456	−28.8%
Model size (KB)	30.69	13.82	−54.98%

For model deployment, the 7857 parameters are divided into the core spiking weights group containing 4608 parameters from N_{conv} and N_{spiking}, and the auxiliary group containing 3249 parameters from the remaining layers. The storage size in Kilobytes is given by

$$\text{Size} = \frac{N_{\text{sp}} \times B_{\text{sp}} + N_{\text{aux}} \times B_{\text{aux}}}{8 \times 1024} \text{ KB} \quad (35)$$

where B_{sp} and B_{aux} are the bitwidths of the two parameter groups respectively. For the full precision float32 baseline where all parameters use $B = 32$ bits,

$$\text{Size}_{\text{float32}} = \frac{7857 \times 32}{8 \times 1024} \approx 30.69 \text{ KB} \quad (36)$$

For the ternary deployment, the core spiking weights are stored at $B_{\text{sp}} = 2$ bits while the auxiliary layers remain at $B_{\text{aux}} = 32$ bits,

$$\text{Size}_{\text{ternary}} = \frac{4608 \times 2 + 3249 \times 32}{8 \times 1024} \approx 13.82 \text{ KB} \quad (37)$$

As shown in Table 6, the Cortex-A53 measurement shows that TTQ reduces software inference cost. Future work will map ternary operations and POPCNT accumulation to FPGA hardware to exploit parallel execution and further reduce latency and energy.

5. Conclusion

PON provides packet-level IoT intrusion detection using 64-byte packet headers, behavior-adaptive redundancy filtering, SAD, and ternary quantization. On the CIC-IoT2023 mixed-stream evaluation, the TTQ+SAD configuration achieves a Window Macro F1 of 92.48% and a detection rate of 90.23%, while reducing model storage from 30.69 KB to 13.82 KB. PON performs header-only inference; payload entropy, when enabled, is used only by the redundancy filter before the packet-retention decision and is not forwarded to the classifier. The 13.82-KB representation remains mixed precision because the input projection and auxiliary operations are float32, so the storage reduction does not imply equivalent latency or energy savings. The mixed-stream construction is a controlled approximation rather than a complete reproduction of live IoT traffic. Future work will validate the method on naturally mixed IoT traffic, quantize the remaining float32 components, and map ternary and POPCNT operations to FPGA accelerators.

Acknowledgements

The authors acknowledge the support of time and facilities from Ho Chi Minh City University of Technology, HCMUT, VNU HCM for this study.

References

[1] Schmitt M. Securing the digital world: Protecting smart infrastructures and digital industries with artificial intelligence (AI)-enabled malware and intrusion detection. Journal of Industrial Information Integration. 2023 Dec;36:100520.

[2] Ferrag MA, Friha O, Hamouda D, Maglaras L, Janicke H. Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning. IEEE Access. 2022;10:40281–40306.

[3] Neto ECP, Dadkhah S, Ferreira R, Zohourian A, Lu R, Ghorbani AA. CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment. Sensors. 2023 June;23(13):5941.

[4] Berman DS, Buczak AL, Chavis JS, Corbett CL. A Survey of Deep Learning Methods for Cyber Security. Information. 2019 Apr;10(4):122.

[5] Khraisat A, Gondal I, Vamplew P, Kamruzzaman J. Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity. 2019;2(1).

[6] Wang Z, Chen H, Yang S, Luo X, Li D, Wang J. A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization. PeerJ Computer Science. 2023 Sept;9:e1569.

[7] Abbas S, Bouazzi I, Ojo S, Al Hejaili A, Sampedro GA, Almadhor A, et al. Evaluating deep learning variants for cyber-attacks detection and multi-class classification in IoT networks. PeerJ Computer Science. 2024 Jan;10:e1793.

[8] Djaidja TET, Brik B, Mohammed Senouci S, Boualouache A, Ghamri-Doudane Y. Early Network Intrusion Detection Enabled by Attention Mechanisms and RNNs. IEEE Transactions on Information Forensics and Security. 2024;19:7783–7793.

[9] Kim T, Pak W. Early Detection of Network Intrusions Using a GAN-Based One-Class Classifier. IEEE Access. 2022;10:119357–119367.

[10] Becerra-Suarez FL, Tuesta-Monteza VA, Mejia-Cabrera HI, Arcila-Diaz J. Performance Evaluation of Deep Learning Models for Classifying Cybersecurity Attacks in IoT Networks. Informatics. 2024;11(2):32.

[11] Tseng SM, Wang YQ, Wang YC. Multi-Class Intrusion Detection Based on Transformer for IoT Networks Using CIC-IoT-2023 Dataset. Future Internet. 2024;16(8):284.

[12] Sudyana D, Yudha F, Lin YD, Lai CH, Lin PC, Hwang RH. From Flow to Packet: A Unified Machine Learning Approach for Advanced Intrusion Detection. Security and Communication Networks. 2025 Jan;2025(1).

[13] Doriguzzi-Corin R, Knob LAD, Mendozzi L, Siracusa D, Savi M. Introducing packet-level analysis in programmable data planes to advance Network Intrusion Detection. Computer Networks. 2024 Feb;239:110162.

[14] Schuman CD, Kulkarni SR, Parsa M, Mitchell JP, Date P, Kay B. Exploring Neuromorphic Computing Based on Spiking Neural Networks: Algorithms to Hardware. ACM Computing Surveys. 2022;55(4):1-49.

[15] Roy K, Jaiswal A, Panda P. Towards spike-based machine intelligence with neuromorphic computing. Nature. 2019;575(7784):607-17.

[16] Esser SK, Merolla PA, Arthur JV, Cassidy AS, Appuswamy R, Andreopoulos A, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. Proceedings of the National Academy of Sciences. 2016;113(41):11441-6.

[17] Kim S, Park S, Na B, Yoon S. Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection. arXiv preprint arXiv:190306530. 2019.

[18] Chen G, et al. Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip. Nature Communications. 2024;15:4297.

[19] Yamazaki K, Vo-Ho VK, Bulsara D, Le N. Spiking Neural Networks and Their Applications: A Review. Brain Sciences. 2022;12(7):863.

[20] Zhou S, Li X. Spiking Neural Networks with Single-Spike Temporal-Coded Neurons for Network Intrusion Detection. In: 2020 25th International Conference on Pattern Recognition (ICPR). IEEE; 2021. p. 8148-55.

[21] Wang Z, Ghaleb FA, Zainal A, Siraj MM, Lu X. An efficient intrusion detection model based on convolutional spiking neural network. Scientific Reports. 2024 Mar;14(1).

[22] Pawar A, Tiwari N. A Novel Approach of DDOS Attack Classification with Genetic Algorithm-optimized Spiking Neural Network. International Journal of Computer Network and Information Security. 2024 Apr;16(2):103-16.

[23] Zhang J, Zhang M, Wang Y, Liu Q, Yin B, Li H, et al. Spiking Neural Networks With Adaptive Membrane Time Constant for Event-Based Tracking. IEEE Transactions on Image Processing. 2025;34:1009-21.

[24] Balafrej I, Bahadi S, Rouat J, Alibart F. Enhancing temporal learning in recurrent spiking networks for neuromorphic applications. Neuromorphic Computing and Engineering. 2025 May;5(2):024008.

[25] Zhu C, Han S, Mao H, Dally WJ. Trained Ternary Quantization. 2016.

[26] Abbott LF. Lapicque's introduction of the integrate-and-fire model neuron (1907). Brain Research Bulletin. 1999 Nov;50(5-6):303–304.

[27] Burkitt AN. A Review of the Integrate-and-fire Neuron Model: I. Homogeneous Synaptic Input. Biological Cybernetics. 2006 Apr;95(1):1–19.

[28] Neftci EO, Mostafa H, Zenke F. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks. IEEE Signal Processing Magazine. 2019 Nov;36(6):51–63.