Internet of Things 36 (2026) 101879

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

# Transformer-based classification of IoT network traffic with flow-to-window aggregation

Sergio Martin-Reizabal [a,*], Adrian Caballero-Quiroga [a], Beatriz Gil-Arroyo [a], Nuño Basurto [a], Ruben Ruiz-Gonzalez [b]

[a] Grupo de Inteligencia Computacional Aplicada (GICAP), Universidad de Burgos, Av. Cantabria s/n, Burgos, 09006, España
[b] Grupo de Investigación en Automatización, Robótica, Control y Optimización (ARCO), Universidad de Burgos, Avda. Cantabria s/n, Burgos, 09006, España

ARTICLE INFO

*Keywords:*
Internet of things (IoT)
Network traffic classification
Intrusion detection system (IDS)
Transformer
Self-attention
Deep learning

ABSTRACT

The explosive growth of the IoT has led to an increasingly complex and heterogeneous network traffic, posing major challenges for intrusion detection. Most existing machine learning and deep learning approaches model network traffic at the level of individual flows, which limits their ability to capture contextual relationships among concurrent communications. This paper introduces a Transformer-based framework for IoT intrusion detection that aggregates network flows into fixed-duration windows and treats each flow as a token within the input sequence. The self-attention mechanism captures contextual relationships among concurrent flows, enabling effective modeling of temporal dependencies without recurrence. Experiments conducted on the CICIoT2023 dataset show that the proposed model achieves a weighted F1-score of 97.9% and a macro ROC–AUC of 99.6% under temporally blocked cross-validation, while maintaining high computational efficiency. These results demonstrate that flow-to-window aggregation combined with self-attention provides a robust and scalable foundation for IoT network security, suitable for deployment in edge and smart-home environments.

## 1. Introduction

The heterogeneity of Internet of Things (IoT) devices has led to an explosive growth of network traffic in recent years. This traffic shows unique characteristics due to the diversity of devices, protocols, and communication patterns, which makes it hard to apply traditional threat detection and classification methods [1]. The number of IoT devices worldwide is projected to reach 40.6 billion by 2034, more than double the 19.8 billion expected for 2025 [2], many of them running with basic firmware, limited computing resources, and weak security [1]. This situation poses a challenge for Intrusion Detection System (IDS): classical signature or rule-based monitoring lacks the flexibility to detect new behaviors and often requires more processing power than IoT environments can provide. Therefore, IDS are commonly deployed on gateways, routers, edge nodes, or in the cloud, where traffic can be monitored without draining the constrained resources of end devices [3].

In the specific context of Intrusion Detection for IoT environments, real-time traffic classification is the first line of defense. Traditionally, the problem has been approached from two complementary lines. First, the literature includes statistical and “lightweight”

* Corresponding author.
*E-mail addresses:* smreizabal@ubu.es (S. Martin-Reizabal), acquiroga@ubu.es (A. Caballero-Quiroga), bgarroyo@ubu.es (B. Gil-Arroyo), nbasurto@ubu.es (N. Basurto), ruben.ruiz@ubu.es (R. Ruiz-Gonzalez).

https://doi.org/10.1016/j.iot.2026.101879
Received 31 October 2025; Received in revised form 22 December 2025; Accepted 18 January 2026
Available online 21 January 2026

Machine Learning methods (e.g., Support Vector Machine (SVM), Random Forest, $k$–Nearest Neighbors ($k$-NN)) which, despite their low resource needs, process flows in isolation and struggle to capture Temporal Dependency [4]. Second, Deep Learning (DL) solutions (e.g., Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)) improve accuracy and can handle temporal dependencies, but require training and inference costs that are not practical for in situ execution on home gateways or Edge Computing routers [5].

Transformers have recently emerged as an alternative that combine accuracy and temporal dependencies with parallel computation [6]. Unlike Recurrent Models, they use an Attention Mechanism that allows the model to focus more on the most relevant parts of the input when making each prediction. Models without this mechanism, such as Recurrent Neural Network (RNN)s, often struggle to keep track of all the information in long inputs, which can reduce their ability to understand complex patterns over time. However, their use in IoT network security is still not well studied. Existing works mostly focus on single flows and often treat each feature in a flow as an independent token, leaving open both the best level of detail for analysis and how to represent IoT traffic as a structured input suitable for Self-Attention [7,8].

To bridge this gap, this work introduces a novel Flow-to-Window Aggregation framework designed to make IoT network traffic suitable for Transformer-based analysis. In this approach, each network flow is treated as an individual token, and all flows within the same time window are grouped together to form the input sequence for the model. The main contributions of this study are summarized as follows:

1. It proposes a Transformer-based intrusion detection framework for IoT that adopts a flow-to-window formulation, where traffic is organized into fixed-duration temporal windows and each complete network flow is preserved as a token. This design enables the model to capture contextual relationships among concurrent flows, rather than analyzing flows in isolation or relying on aggregated window-level representations.
2. It explores different context window sizes to determine the optimal number of flows to retain within each temporal window for accurate prediction.
3. It applies the proposed framework to a recent IoT dataset [9], demonstrating its effectiveness under a temporally faithful evaluation protocol.

The remainder of this paper is organized as follows. Section 2 contextualizes other existing works in the literature related to our proposed method. Section 3 details our proposed Flow-to-Window Aggregation method and Transformer model architecture. Section 4 explains how the assessment of the proposed method's performance was conducted and the experimental setup. Section 5 presents the results and their discussion. Section 6 discusses the results and compares with existing works in the state of the art. Finally, Section 7 concludes the paper and outlines future research directions.

## 2. Related work

The exponential growth of Internet of Things (IoT) devices has introduced new challenges to network security, as the heterogeneity, scalability, and resource constraints of IoT environments increase their vulnerability to cyberattacks. This section provides an overview of prior work in three key areas: cybersecurity in IoT networks, deep learning for intrusion detection in IoT, and the application of Transformers in network security and IoT.

### 2.1. Cybersecurity in IoT networks

Gyamfi and Jurcut [3] highlight that the rapid proliferation of IoT devices has profoundly transformed modern network infrastructures, introducing new and complex cybersecurity challenges. Similarly, Schiller et al. [10] report that current IoT ecosystems already comprise billions of heterogeneous and interconnected devices, with forecasts anticipating tens of billions more in the near future. This massive growth, as noted by Sharma et al. [11], has significantly expanded the global attack surface and generated unprecedented volumes of network traffic. Many IoT devices remain resource-constrained –with limited CPU, memory, and power– and often rely on outdated or minimal firmware lacking adequate security mechanisms, making them highly vulnerable to exploitation through weak credentials or unpatched firmware vulnerabilities [12]. Zhang [12] further documents the infamous *Mirai* botnet, which compromised millions of IoT devices to conduct large-scale Denial of Service (DDoS attacks, demonstrating the disruptive potential of insecure IoT deployments.

Sharma et al. [11,13] highlighted that compromised IoT nodes in critical sectors such as healthcare, industrial control, and smart grids can disrupt essential services, causing severe consequences. They also noted that the increasing interconnection between cyber and physical systems amplifies the risk of cascading failures in critical infrastructures. Zhang [12] argued that traditional security measures, such as frequent patching or endpoint antivirus software, are often impractical due to device heterogeneity, resource constraints, and large-scale deployments.

Gyamfi and Jurcut [3], Dritsas and Trigka [1], and Zarpelão et al. [14] emphasized that, given the limitations of individual IoT devices, network-based security approaches have become essential. They noted that instead of securing each device separately, the network itself should be continuously monitored to detect anomalies and malicious activity. Schiller et al. [10] and Zarpelão et al. [14] argued that intrusion detection solutions tailored to IoT must consider device constraints, communication protocols, and context-awareness to provide effective protection.

These challenges have motivated the adoption of intelligent IDS based on ML and DL, capable of automatically detecting complex and evolving attack patterns in IoT environments.

### 2.2. Deep learning for intrusion detection in IoT

Given the limitations of traditional IDS in handling high-dimensional, heterogeneous, and real-time IoT traffic, DL architectures have emerged as a promising solution, as they can automatically extract complex features from network flows and adapt to evolving attack patterns, offering superior performance compared to classical ML approaches.

Altunay and Albayrak [15] and Wang et al. [16] report that early deep learning approaches for IoT security primarily relied on CNNs to learn discriminative spatial features from network flow statistics, while Nazir et al. [17] highlighted that RNNs, particularly LSTM and GRU variants, were later introduced to capture temporal dependencies in sequential data.

Alsubaei et al. [18] proposed an Optimized Sequential Neural Network (OSNN) composed of convolutional and fully connected layers for IoT intrusion detection, achieving accuracies above 99% across benchmark datasets through extensive hyperparameter optimization.

Adefemi et al. [19] demonstrated that hybrid CNN–GRU architectures can effectively integrate spatial and temporal information, achieving 99.83% accuracy, 99.82% recall, and an F1-score of 99.83% on the IoTID20 dataset for IoT intrusion detection (IoTID20) dataset [20], and 99.01% accuracy with an AUC of 0.99 on BoT-IoT dataset for IoT botnet traffic (BoT-IoT) [21].

Khan et al. [22] demonstrated that their hybrid RNN–GRU multi-layer model achieves 99% accuracy on network flow datasets and 98% on application layer datasets of the ToN-IoT dataset [23], outperforming previous deep learning and traditional approaches. Meanwhile, Sasi et al. [24] developed a 1D-CNN–LSTM model with a self-attention mechanism, reporting accuracies of up to 99.96% on tabular datasets constructed from multiple IoT traffic traces. These traces were collected from different sources and subsequently preprocessed into flow-level features using CICFlowMeter, resulting in a unified tabular representation rather than a single native dataset. Similarly, Alashjaee [25] proposed an Attention-CNN-LSTM model, which obtained 94.8% accuracy on the NSL-KDD dataset (NSL-KDD) [26], and 97.5% accuracy on Bot-IoT, confirming that incorporating self-attention mechanisms significantly improves performance in IoT intrusion detection.

Khan et al. [22] further showed that a multi-layer RNN–GRU model provides superior performance compared to single-paradigm models, while Sasi et al. [24] and Alashjaee [25] highlighted the benefits of incorporating self-attention into CNN–LSTM frameworks.

Hossain et al. [27] argue that pure Reinforcement learning based intrusion detection systems are better suited for highly dynamic and evolving threat environments than static deep learning classifiers. In particular, their Deep Q-learning Intrusion Detection System (DQ-IDS) formulates intrusion detection as a sequential decision-making problem and relies exclusively on Deep Q-Learning to achieve adaptive and self-learning behavior, without incorporating CNN, LSTM, or hybrid deep architectures, reporting a detection accuracy of 97.18% when evaluated on the NSL-KDD dataset, while explicitly emphasizing adaptability to unseen attacks. In a separate and independent study, Hossain performs a comparative evaluation of several static deep learning models for intrusion detection, including 1D-CNN, LSTM, RNN, and MLP, and concludes that a 1D convolutional neural network consistently outperforms recurrent and hybrid architectures in both binary and multiclass settings, achieving up to 99.53% accuracy on the same dataset. Notably, this performance gain is attributed to architectural suitability and hyperparameter optimization rather than the use of more complex or hybrid network designs [28].

Other studies have targeted specific threats, such as Man-in-the-Middle (MitM) attacks, reporting detection accuracies above 94% [29], while Federated Learning (FL) and Knowledge Distillation (KD) approaches have been explored to enhance privacy and generalization across heterogeneous IoT environments [30,31].

The adoption of Explainable Artificial Intelligence (XAI) techniques has been investigated by Sharma et al. [32] to interpret model decisions and foster trust in critical systems, and Serrano [33] stressed that deployment at gateways or edge environments requires lightweight, efficient models capable of operating in distributed IoT networks.

While hybrid CNN–RNN models, such as those proposed by Adefemi et al. [19] and Alsubaei [18], achieve remarkable detection accuracies, their evaluations remain largely confined to flow-level analysis. As a result, these architectures may not fully capture inter-flow dependencies or broader contextual behaviors across simultaneous IoT communications–an aspect that warrants further investigation in future research. Nazir et al. [34] further highlighted that ML and DL models for IoT intrusion detection often face difficulties in maintaining generalization across heterogeneous IoT environments. Meanwhile, Keshk et al. [35] underscored the importance of incorporating explainability into deep learning-based IDS, noting that interpretability in cybersecurity applications remains underexplored. Collectively, these studies motivate ongoing research into architectures–such as Transformer-based models–that can better capture global relationships across flows and offer more transparent, context-aware intrusion detection.

### 2.3. Transformers in network security and IoT

Recent work has explored applying transformers for IoT network security because self-attention can capture long-range dependencies while scaling well to heterogeneous, mixed-type inputs. Within this line, four representative directions –tokenization at the feature-value level, tabular per-flow modeling, window-level embedding, and multimodal fusion– have emerged.

Afifi et al. [36] introduced MIND-IoT, a hybrid Transformer–CNN framework that serializes each flow into an ordered sequence of feature-value pairs using a domain-specific tokenizer (IoT-Tokenize). Feature names are preserved as predefined tokens, while values are segmented using byte-level Byte-Pair Encoding (BPE); the encoder is pre-trained with masked language modeling, and fine-tuning employs a lightweight CNN head over the token embeddings. On the CICIoT2023 dataset [37], MIND-IoT achieved 99.62% accuracy, 99.61% recall, and an F1-score of 99.60%, demonstrating strong generalization across IoT environments.

Bazaluk et al. [38] proposed an IoT traffic classification Transformer built on TabTransformer for Message Queuing Telemetry Transport (MQTT) traffic. The model treats each record as a tabular flow instance, embedding categorical columns and integrating

numerical features, with pre-training on a large MQTT dataset [39] and fine-tuning on smaller labeled subsets. It reported an overall accuracy of 82%, addressing data scarcity in MQTT-based scenarios but without temporal aggregation.

Earlier work by Kozik et al. [40] explored the use of Transformer architectures for IoT traffic analysis through a time-window embedding strategy. In their approach, network flows are first summarized within fixed-length temporal windows using statistical descriptors and probabilistic summaries, producing a single embedding per window. Sequences of such window-level embeddings are then processed by a Transformer encoder for classification. While this framework demonstrated the feasibility of applying self-attention to temporally segmented IoT traffic, it relies on window-level statistical aggregation and does not explicitly model interactions among individual concurrent flows within the same window. In this setting, the Transformer operates over sequences of consecutive window embeddings, capturing temporal dependencies across windows rather than relationships among concurrent flows within a single window.

Wang et al. [41] developed an FT-Transformer-based Network Intrusion Detection System (NIDS) for smart homes that fuses network-flow statistics with IoT sensor telemetry. The self-attention encoder learned contextual embeddings across heterogeneous categorical and numerical features, achieving 98.39% accuracy on the Telemetry of Networked IoT dataset (ToN-IoT) dataset.

Tseng et al. [42] designed a Transformer-only model for multi-class intrusion detection using CICIoT2023 dataset, enhancing encoder normalization and residual depth. The model achieved 99.7% accuracy, 99.6% recall, and 99.6% F1-score across ten attack types. Zhang et al. [43] further proposed a hybrid CNN–Bidirectional Long Short-Term Memory (BiLSTM)–Transformer architecture, where CNN layers extract local spatial patterns, BiLSTM units capture bidirectional intra-flow dependencies, and the Transformer refines global relationships via multi-head attention. Their model achieved 99.80% accuracy, 99.94% recall, and 99.81% precision on the Canadian Institute for Cybersecurity IDS 2017 dataset (CIC-IDS2017) dataset.

To provide a structured comparison of the discussed literature, Table 1 summarizes the key methodologies, architectural choices, and reported performance metrics of the representative studies reviewed in this section.

Although these studies achieve state-of-the-art metrics through deeper architectures, enhanced normalization, or multimodal feature fusion, they largely operate either at the *per-flow level* or on *aggregated window-level representations*. In both cases, network flows are treated as independent instances or collapsed into summary embeddings, which limits the ability to explicitly model temporal relationships and interactions among multiple concurrent flows. As a result, the potential of self-attention mechanisms to capture inter-flow contextual dependencies remains underexploited in existing intrusion detection approaches.

*Positioning of this study.* In contrast, this study represents each temporal window as a structured set of concurrent flows, where each flow is treated as a token and the Transformer processes a bounded sequence of flows corresponding to a fixed-duration window. In this formulation, self-attention operates directly across flows, capturing concurrency, burstiness, and inter-flow dependencies, and providing the conceptual foundation for the proposed architecture and experimental evaluation.

## 3. Proposed method

This section presents the proposed end-to-end pipeline for classifying IoT network activity within short time windows using a Transformer encoder. The overall workflow, illustrated in Fig. 1, is divided into four main stages. First, raw Packet CAPture (PCAP) files are processed to extract per-flow statistics using CICFlowMeter (Section 3.1). Next, these flows are aggregated into fixed-duration temporal windows to capture contextual relationships among concurrent communications (Section 3.2). Each resulting window is then subjected to Tensorization and feature normalization to produce a standardized, Masked sequence suitable for model input (Section 3.3). Finally, a lightweight Transformer encoder processes the masked flow sequences with self-attention to generate one label per window (Section 3.4).

### 3.1. Flow extraction from PCAP traces

PCAP traces are converted into bidirectional flows with CICFlowMeter [44] tool, which establishes forward/backward directions of the Five-Tuple (SrcIP, DstIP, SrcPort, DstPort, Protocol) from the first observed packet and computes more than 80 per-flow features. Flow delimitation is given by two timeouts:

- **Flow Timeout (idle timeout): 10 s.** If no packets are observed for a given five-tuple during 10 s, the flow is closed (particularly relevant for User Datagram Protocol (UDP), which lacks an explicit FIN Signal signal in contrast to Transmission Control Protocol (TCP)).
- **Activity timeout: 5 s.** The maximum active duration of a flow is capped at 5 s even if packets keep arriving, preventing long-lived aggregates per-flow (Activity Timeout).

These values, combined with $W = 5$ s windows (see Section 3.2), ensure that a single flow appears in, at most, *two* consecutive windows and avoid same flows of spanning in many consecutive windows. Each exported Comma- Separated Values (CSV) corresponds to a single traffic condition (benign or one attack type), and its label is propagated to all rows.

### 3.2. Windowing

Intuitively, the windowing process aggregates network activity into a sequence of fixed-duration temporal windows, each intended to capture a coherent snapshot of concurrent flows. Rather than truncating flows at window boundaries, each flow is assigned to

**Table 1**
Summary of representative IDS literature reviewed. "Instance" denotes the atomic granularity used for classification. Metrics are reported as stated in the corresponding papers (**bin**: binary classification; **mc**: multi-class classification). The last row corresponds to this work.

| Work (Year) | Instance | Model | Datasets | Key reported metrics (bin/mc) |
|---|---|---|---|---|
| Altunay & Albayrak (2023) [15] | Flow | CNN–LSTM | UNSW-NB15<br>X-IIoTID | Acc 93.21% (bin) / 92.9% (mc)<br>Acc 99.84% (bin) / 99.8% (mc) |
| Wang et al. (2023) [16] | Flow | ResNet + Transformer + BiLSTM | NSL-KDD<br>CICIDS2017<br>MQTTset | Acc 90.99% (bin)<br>Acc 99.15% (bin)<br>Acc 99.56% (bin) |
| Nazir et al. (2024) [17] | Flow | CNN–LSTM (+ PCA) | IoT-23<br>N-BaIoT<br>CICIDS2017 | Acc 95% (bin)<br>Acc 99% (bin)<br>Acc 99% (bin) |
| Alsubaei et al. (2025) [18] | Flow | OSNN (Tuned) | NSL-KDD<br>UNSWNB15 | Acc 99.93% (mc),<br>F1 99.8% |
| Adefemi et al. (2025) [19] | Flow | CNN–GRU | IoTID20<br>BoT-IoT | Acc/F1 99.83% (bin)<br>Acc 99.01% (bin) |
| Khan et al. (2023) [22] | Flow | RNN–GRU | ToN-IoT (Net)<br>ToN-IoT (App) | Acc 99% (bin)<br>Acc 98% (bin) |
| Sasi et al. (2024) [24] | Flow | 1D-CNN–LSTM + Self-Attn | IoT flows<br>(CICFlowMeter) | Acc ≈ 99.96% (bin) |
| Alashjaee et al. (2025) [25] | Flow | Attn-CNN–LSTM | BoT-IoT<br>NSL-KDD | Acc 97.5% (mc)<br>Acc 94.8% (mc) |
| Afifi et al. (2024) [36] | Flow | Transformer (+ Tokenizer) | CICIoT2023 | Acc 99.62%, F1 99.60% (bin) |
| Wang et al. (2023) [41] | Flow | FT-Transformer | ToN-IoT | Acc 98.39% (bin) |
| Bazaluk et al. (2023) [38] | Flow | TabTransformer (MQTT) | MQTT-IoT-IDS2020 | Acc 82% (bin) |
| Tseng et al. (2024) [42] | Flow | Transformer Encoder | CICIoT2023 | Acc 99.7% (mc), F1 99.6% |
| Zhang et al. (2025) [43] | Flow | CNN + BiLSTM + Transformer | CIC-IDS2017<br>BoT-IoT | Acc 99.80% (bin)<br>Acc 97.95% (bin) |
| Hossain (2025) [27] | Flow | Deep Q-Network (RL) | NSL-KDD | Acc 97.18% (bin) |
| Kozik et al. (2021) [40] | Sequence of window embeddings | Transformer Encoder | IoT-23 | Acc 99%, F1 99% (bin) |
| **This work** | **Window (Flows as tokens)** | **Transformer Encoder + [CLS]** | CICIoT2023 | **Bal Acc 97.8% (mc), AUC macro 99.6 (mc)** |

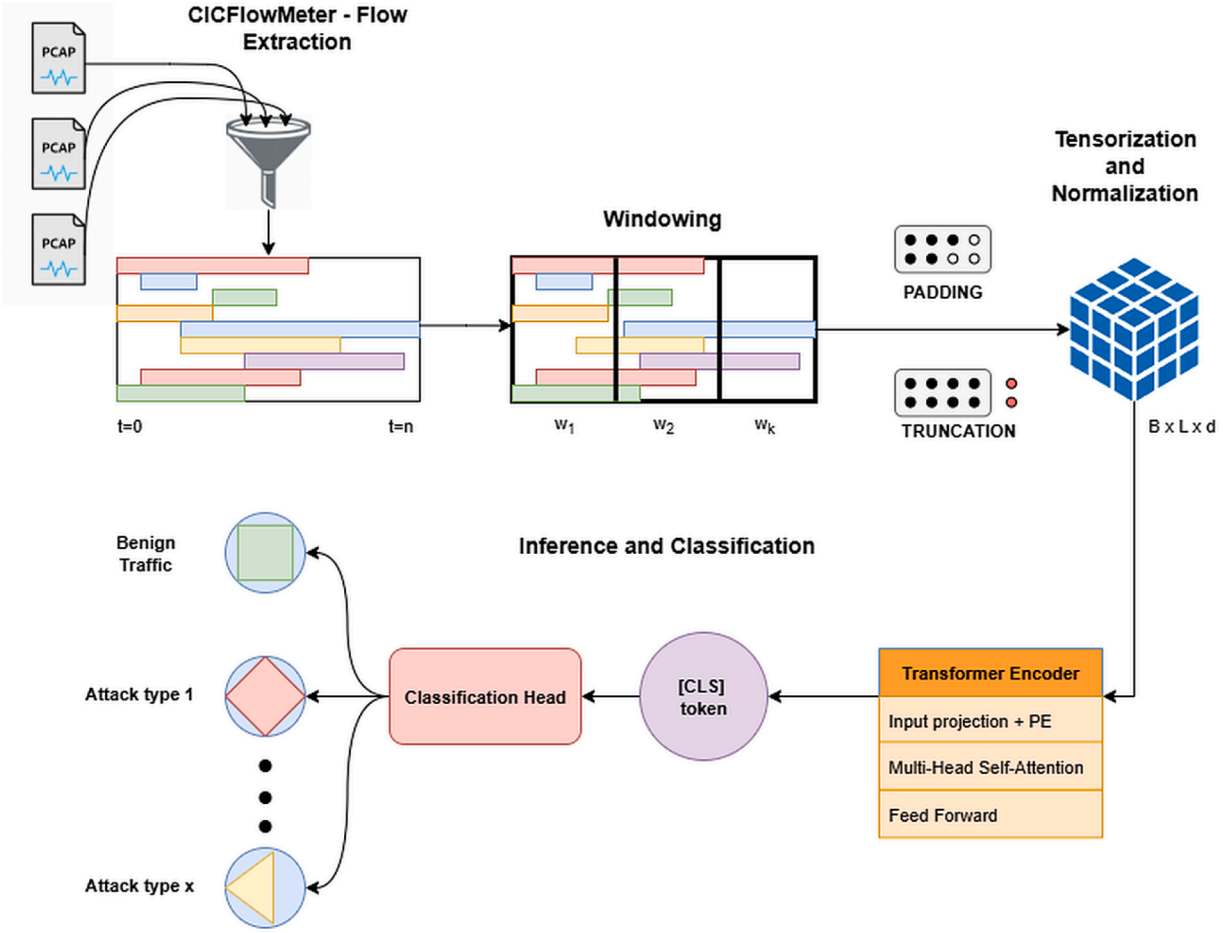


**Fig. 1.** End-to-end pipeline overview. Raw PCAP traces are converted into bidirectional flows (CICFlowMeter, timeouts) and aggregated into fixed-duration windows ($W = 5$ s). Each window is then tensorized by selecting numeric features, handling missing/invalid values, applying feature standardization, and performing padding/truncation with an associated mask. Finally, the Transformer module projects the standardized per-flow feature vectors to the model dimension, prepends a learned `[CLS]` token, applies masked self-attention over the resulting flow sequence, and outputs a single class label per window.

all windows that overlap its active time interval. This ensures that long-lived flows spanning multiple windows are consistently represented, while each window reflects the set of flows that were active during that time span.

Given a window size of $W$ s, each flow with start time $t_s$ (in seconds) and duration $\Delta_{\mu s}$ (in microseconds) ends at $t_e$, computed as in Eq. (1):

$$t_e = t_s + \frac{\Delta_{\mu s}}{10^6}. \tag{1}$$

A flow is assigned to every half-open window $[kW, (k+1)W)$ that overlaps its active interval $[t_s, t_e)$, i.e., it is included whenever the overlap condition in Eq. (2) holds:

$$[kW, (k+1)W) \cap [t_s, t_e) \neq \varnothing : k \in \mathbb{N}_0. \tag{2}$$

To avoid boundary artifacts, a small $\epsilon > 0$ is used and the first/last window indices are computed as in Eq. (3):

$$k_0 = \left\lfloor \frac{t_s}{W} \right\rfloor, \qquad k_1 = \left\lfloor \frac{t_e - \epsilon}{W} \right\rfloor. \tag{3}$$

The procedure then emits one row for each $k \in \{k_0, \ldots, k_1\}$ annotated with the key `Window_Start` $= kW$, as implied by Eq. (3). Note that, by the overlap condition in Eq. (2), if no flow intersects a given window, no row is created for that $k$; in other words, empty windows are not considered.

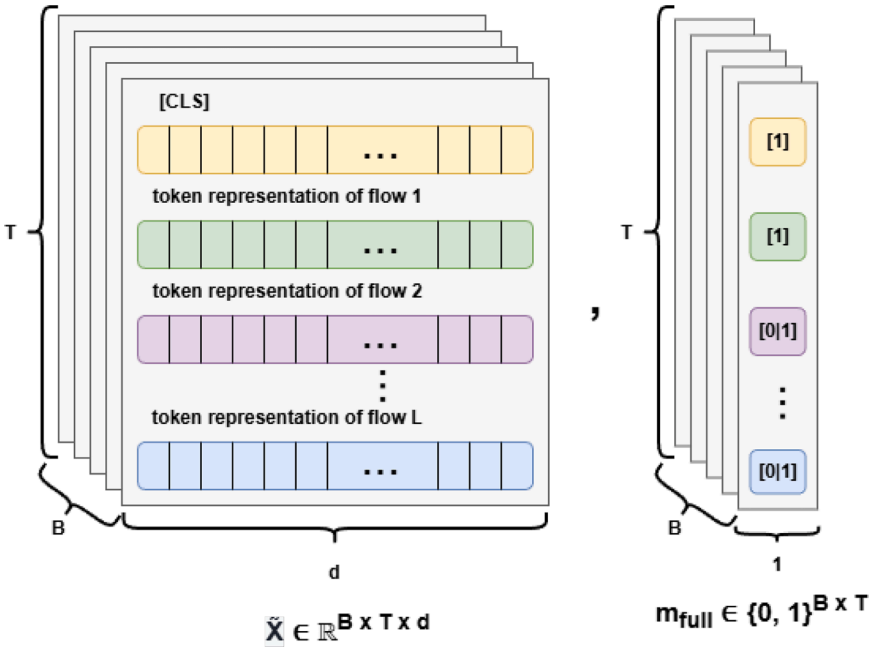


**Fig. 2.** Input representation for a batch of windows. For each window, the sequence begins with a learned `[CLS]` token followed by a fixed-length sequence of L flow tokens (one token per flow), adding up to $T = L + 1$ tokens. Flows are ordered by starting time of each flow within the window. The accompanying binary mask $m_{\text{full}}$ marks real tokens as 1 and padding positions as 0, and it is this mask that determines which positions are ignored by masked self-attention. $B$ denotes the size of the batch and $d$ is the latent space dimension after the embedding.

### 3.3. Tensorization and feature normalization

For each window, only numeric features are retained (identifiers such as `Flow ID`, IP addresses, ports, protocol, and `Timestamp` are discarded). NaN and $\pm\infty$ values are replaced by `0.0`, and flows are sorted by `Timestamp`. Let $d_{\text{in}}$ denote the number of retained features per flow. A window with $n$ flows yields a matrix $X \in \mathbb{R}^{n \times d_{\text{in}}}$.

To batch variable-length windows, a maximum number of flow tokens per window is fixed as $L$. At most $L$ flows are kept per window (by truncation if $n > L$, or by zero-padding if $n < L$). Padding is applied after the last real flow so that chronological order is preserved. A boolean mask, $mask \in \{0, 1\}^L$, marks valid positions (1) and padding (0).

Features are standardized using a StandardScaler (zero mean, unit variance) fit exclusively on the training windows and reused for validation/test. Note that the `[CLS]` token is not added at this stage: it is prepended later inside the Transformer module after the input projection to the model dimension (see Section 3.4).

The resulting padded and standardized flow sequences, together with their corresponding masks, constitute the direct input to the Transformer encoder described next.

### 3.4. Transformer encoder with masked self-attention

Let $B$ be the batch size, $L$ the number of real flow tokens per window (after truncation/padding), $T = L + 1$ the total length including the `[CLS]` token, $d_{\text{in}}$ the input feature size per flow, $d$ the embedding model dimension, $H$ the number of heads, $d_h = d/H$ the head size, $d_{ff}$ the hidden dimension of the feed-forward network, $N$ the number of encoder layers and $C$ the number of classes.

Figs. 2 and 3 jointly illustrate how the proposed Transformer receives and processes windowed traffic. Fig. 2 shows the *input sequence* built for each window: a token sequence that starts with a learned `[CLS]` token followed by up to $L$ flow tokens (one per flow), where missing positions are filled with padding. In parallel, a binary padding mask is provided so that padded positions are ignored by attention. Formally, the batched input sequence is $\tilde{X} \in \mathbb{R}^{B \times T \times d}$ (with $T = L + 1$). In this representation, tokens are already expressed in the embedding model dimension $d$, i.e., after the input projection (*Input Embedding*) shown in Fig. 3. The corresponding padding mask is $m_{\text{full}} \in \{0, 1\}^{B \times T}$, where values of 1 indicate positions containing real flow tokens (including `[CLS]`) and values of 0 indicate padding positions; masked self-attention uses this mask to ignore padded flows. Fig. 3 depicts a single encoder *layer* that is stacked $N$ times: masked multi-head self-attention splits the representation into $H$ heads of size $d_h = d/H$ (see Eqs. (12)–(15)), and the position-wise FFN uses hidden size $d_{ff}$ (Eq. (17)). Residual Add&Norm preserves the sequence shape after both sub-blocks, and after $N$ layers the final `[CLS]` state is used for window-level classification (Eqs. (18)-(19)).

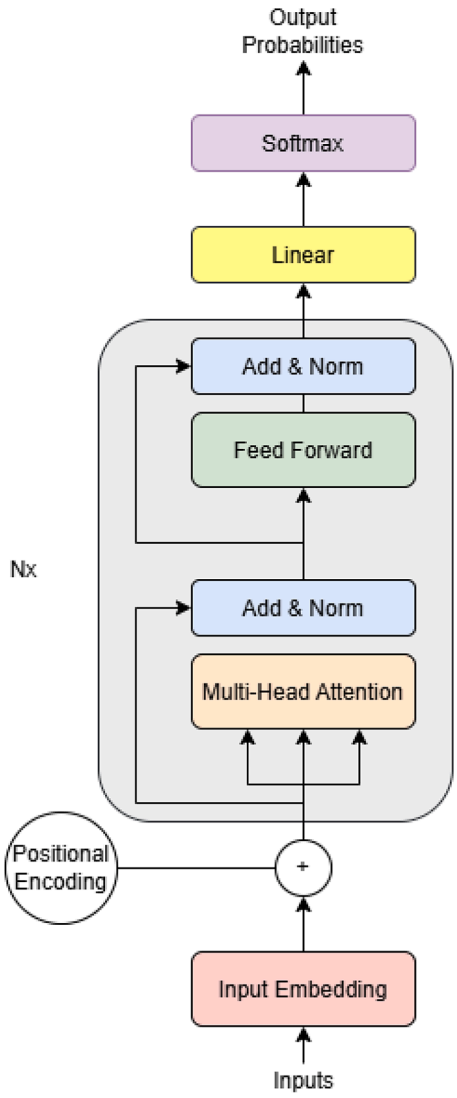


**Fig. 3.** Transformer encoder layer used in this work (stacked $N$ times). Each layer applies masked multi-head self-attention to model interactions among tokens within the same window, followed by residual Add&Norm and a position-wise feed-forward network. The final `[CLS]` embedding acts as a global window summary and is forwarded to the classifier to produce window-level predictions.

### 3.4.1. Dataset and batching

WindowDataset delivers, for each sample (i.e., each window), the triples `(scaled, mask, lbl)`. Each component has the following meaning:

- `scaled` $\in \mathbb{R}^{L\times d_{\text{in}}}$: matrix with the scaled *numerical features* of the flows falling into the window. Rows correspond to flows (chronologically ordered by `Timestamp` of the start of the flow), and columns correspond to features (e.g., CICFlowMeter statistics).
- `mask` $\in \{0,1\}^{L}$: binary vector indicating which positions in `scaled` are *valid* (1) and which correspond to *padding* (0).
- `lbl` $\in \{0,\dots,C-1\}$: integer label of the attack within this window (the classification target).

After batching, the resulting tensor shapes are given in Eq. (4):

$$X \in \mathbb{R}^{B\times L\times d_{\text{in}}}, \qquad m \in \{0,1\}^{B\times L}. \tag{4}$$

### 3.4.2. Input projection and [CLS] token

The tensor $X$ contains $B$ windows, each represented by $L$ flows with $d_{\text{in}}$ standardized features. Since Transformer layers operate in a common latent space of dimension $d$, each flow vector is projected to the embedding model dimension $d$ using a trainable linear

layer. Eq. (5) defines this input projection:

$$X_{\text{proj}} = XW_{\text{in}} + b_{\text{in}}, \qquad X_{\text{proj}} \in \mathbb{R}^{B\times L\times d}. \tag{5}$$

$W_{\text{in}} \in \mathbb{R}^{d_{in}\times d}$ and $b_{\text{in}} \in \mathbb{R}^{d}$ represent the weight matrix and bias vector, respectively, taking into account the necessary broadcasting for proper tensor operations to make sense.

In addition, a special learned token `[CLS]` is introduced to act as a global summary of the window. This token is replicated across the batch dimension (Eq. (6)) and concatenated with the projected flows (Eq. (7)):

$$C_{\text{cls}} \in \mathbb{R}^{B\times 1\times d}. \tag{6}$$

$$\tilde{X} = \left[C_{\text{cls}}\ ;\ X_{\text{proj}}\right] \in \mathbb{R}^{B\times T\times d}, \qquad T = L + 1. \tag{7}$$

### 3.4.3. Mask construction

The boolean mask is extended with a valid position for `[CLS]` as formalized in Eq. (8):

$$m_{\text{full}} = \left[\mathbf{1}_{B\times 1}\ ;\ m\right] \in \{0,1\}^{B\times T}. \tag{8}$$

For attention computation, the mask is reshaped to enable broadcasting over heads and queries. Eq. (9) specifies this transformation:

$$\mathcal{M} \in \{0,1\}^{B\times 1\times 1\times T} \quad \Rightarrow \quad \mathcal{M} \text{ broadcasts to } (B\times H\times T\times T). \tag{9}$$

### 3.4.4. Positional encoding

Since self-attention alone does not encode the order of tokens, explicit positional information must be added so that the model can distinguish the temporal order of flows within each window. To this end, a standard sinusoidal positional encoding is added element-wise to the input token embeddings. Each position in the sequence (including the `[CLS]` token and the flow tokens ordered by time) is associated with a deterministic vector that encodes its relative position in the window. This operation preserves the original tensor shape while enabling the model to distinguish early from late flows within the same temporal window. Formally, positional encodings (PE) are added as expressed in Eq. (10):

$$\tilde{X} \leftarrow \tilde{X} + \text{PE} \in \mathbb{R}^{B\times T\times d}. \tag{10}$$

### 3.4.5. Multi-head attention (per layer)

Given the input sequence $\tilde{X}$, queries ($Q$), keys ($K$), and values ($V$) are obtained through linear projections using the originally proposed mechanism [6]. Eq. (11) defines this operation:

$$Q = \tilde{X}W_Q,\ K = \tilde{X}W_K,\ V = \tilde{X}W_V \in \mathbb{R}^{B\times T\times d}. \tag{11}$$

These tensors are reshaped into $H$ heads of size $d_h = d/H$, according to Eq. (12).

$$Q, K, V\ \rightarrow\ \mathbb{R}^{B\times H\times T\times d_h}. \tag{12}$$

For each head, similarity scores are computed as scaled dot products (Eq. (13)):

$$S = \frac{QK^{\top}}{\sqrt{d_h}} \in \mathbb{R}^{B\times H\times T\times T}. \tag{13}$$

After applying the additive mask from Eq. (9), attention weights are obtained via softmax as shown in Eq. (14).

$$A = \text{softmax}(S) \in \mathbb{R}^{B\times H\times T\times T}. \tag{14}$$

The context is obtained by weighting the values (Eq. (15)):

$$U = A\,V \in \mathbb{R}^{B\times H\times T\times d_h}. \tag{15}$$

Eqs. (11)–(15) define the per-head scaled dot-product attention, which is summarized schematically in Fig. 4.

Finally, all heads are concatenated and linearly projected back to the model dimension (Eq. (16)):

$$Y = \text{Linear}(U) \in \mathbb{R}^{B\times T\times d}. \tag{16}$$

### 3.4.6. Position-wise FFN (per layer)

The output of multi-head self-attention, denoted $Y$, is passed through a position-wise Feed-Forward Network (FFN) applied independently to each token (Eq. (17)):

$$Z = \text{FFN}(Y) = W_2\,\sigma(W_1 Y + b_1) + b_2, \tag{17}$$

with $W_1 \in \mathbb{R}^{d\times d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff}\times d}$ and $\sigma(\cdot)$ being the ReLU activation function.

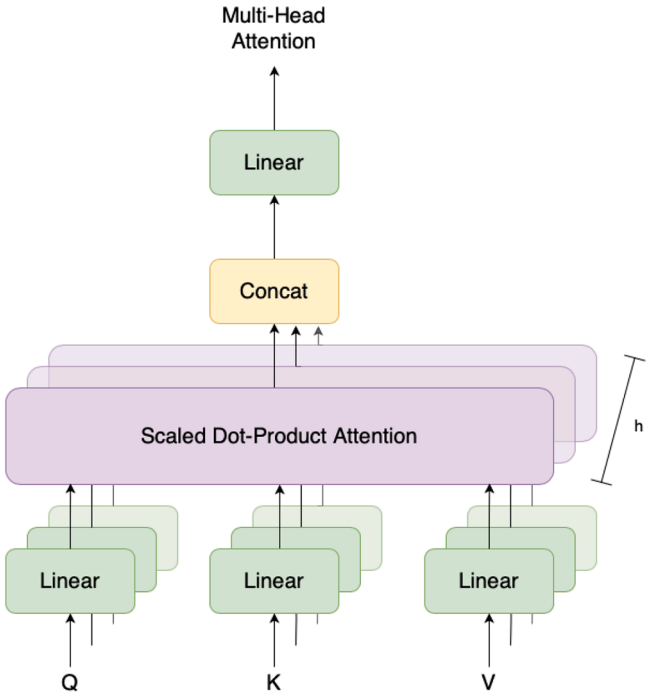


**Fig. 4.** Multi-head self-attention: per-head scaled dot-product attention over $Q, K, V$; heads are concatenated and projected back to model dimension $d$.

**Table 2**
Main tensors and shapes of the proposed model.

| Stage | Shape |
|---|---|
| Batch input $X$ | $B \times L \times d_{\text{in}}$ |
| Mask $m$ | $B \times L$ (1 valid, 0 padding) |
| After $W_{\text{in}}$ | $B \times L \times d$ |
| Concat [CLS] | $B \times T \times d$ $(T = L + 1)$ |
| Attn. mask $\mathcal{M}$ | $B \times 1 \times 1 \times T$ |
| $Q, K, V$ (split heads) | $B \times H \times T \times d_h$ |
| Scores $S$ | $B \times H \times T \times T$ |
| Context (concat heads) | $B \times T \times d$ |
| Encoder output | $B \times T \times d$ |
| [CLS] pooling $z$ | $B \times d$ |
| Logits $\hat{y}$ | $B \times C$ |

### 3.4.7. *Classification from [CLS] token*

After stacking $N$ encoder layers, the final sequence representation is obtained. The embedding corresponding to the [CLS] token is extracted as defined in Eq. (18):

$$z = Z^{(N)}[:, 0, :] \in \mathbb{R}^{B \times d}. \tag{18}$$

This global representation is mapped to class logits through a linear layer, as specified in Eq. (19):

$$\hat{y} = z\, W_c + b_c \in \mathbb{R}^{B \times C}. \tag{19}$$

A summary of the main tensors and their dimensions across the different stages of the Transformer encoder is provided in Table 2.

## 4. Materials and methods

This section describes the materials and methodological framework employed in the study. First, the dataset used for experimentation is presented, including its composition and preprocessing steps. The temporal cross-validation protocol is then detailed to ensure a realistic evaluation under time-dependent conditions. The subsequent subsection outlines the preliminary hyperparameter selection process, focusing specifically on the variation of the context window size in order to identify the optimal configuration for model performance. The model architecture and training setup adopted in this work are then described. Finally, the metrics and reporting procedures used to assess and compare the models are specified, along with the hardware configuration of the system used for experimentation.

### 4.1. Dataset

The experiments to assess the performance of the proposed model were conducted on the Canadian Institute for Cybersecurity IoT 2023 dataset (CICIoT2023) dataset [9], which comprises 33 distinct attacks organized into seven families: DDoS, DoS, Reconnaissance, Web-based, Brute Force, Spoofing, and Mirai. Each scenario was captured as a PCAP file and later converted into bidirectional flow records using CICFlowMeter [44], which extracts over 80 statistical features per flow. Based on these flows, 5-s temporal windows were generated to serve as the main classification units, producing a set of *window-level* CSV files. Files with the suffix `_windows.csv` correspond to 5-s aggregated windows built from flow-level data.

While the original dataset includes 33 attacks, the study focused on a representative subset including both benign and malicious traffic, in order to make it easier to evaluate the performance metrics. The selected attacks, chosen to cover volumetric and reconnaissance behaviors, were as follows: *BENIGN*, *DDoS-ICMP*, *DoS-TCP*, *DDoS-TCP*, OSScan, and PortScan. The corresponding window-level CSV files employed in the experiments are listed below:

```
csv_paths = {
"BENIGN": ["./PCAPS/Benign_0_windows.csv",
"./PCAPS/Benign_1_windows.csv",
"./PCAPS/Benign_2_windows.csv",
"./PCAPS/Benign_3_windows.csv"],
"DDoS-ICMP": "./PCAPS/DDoS-ICMP_0_windows.csv",
"DoS-TCP": ["./PCAPS/DoS-TCP-0_00000_windows.csv",
"./PCAPS/DoS-TCP-0_00004_windows.csv",
"./PCAPS/DoS-TCP-1_00000_windows.csv",
"./PCAPS/DoS-TCP-1_00002_windows.csv",
"./PCAPS/DoS-TCP-2_00000_windows.csv",
"./PCAPS/DoS-TCP-2_00001_windows.csv",
"./PCAPS/DoS-TCP-2_00002_windows.csv",
"./PCAPS/DoS-TCP-2_00004_windows.csv"],
"DDoS-TCP": ["./PCAPS/DDoS-TCP-0_windows.csv",
"./PCAPS/DDoS-TCP-1_windows.csv",
"./PCAPS/DDoS-TCP-2_windows.csv",
"./PCAPS/DDoS-TCP-3_windows.csv"],
"OSScan": "./PCAPS/Recon-OSScan_windows.csv",
"PortScan": "./PCAPS/Recon-PortScan_windows.csv"
}
```

In certain attack scenarios, the original PCAPs contained a very large number of concurrent flows. To prevent memory saturation during processing, some of these captures were divided into smaller sub-traces (e.g., `0_00000`, `0_00004`), each representing a temporal segment of the same attack instance.

Table 3 presents descriptive statistics of the number of windows derived from each processed trace, together with the distribution of flow counts aggregated within each 5-s window. These statistics highlight the strong variability in traffic density across benign and attack conditions, particularly for volumetric DoS and DDoS scenarios.

### 4.2. Temporal cross-validation protocol

To obtain temporally faithful estimates while ensuring that every split contains all labels, the evaluation follows a label-wise stratified temporal Cross-Validation (CV) scheme with adjacency blocking, hereafter referred to as Temporally blocked cross-validation. The protocol operates on *unique windows* and proceeds in three stages.

*(i) Label-wise folding by windows.* For each label $\ell$ independently, the set of unique windows $\{w\}$ that carry $\ell$ is sorted by start time and partitioned into $n_{\text{folds}}$=100 *contiguous* folds with approximately equal number of windows, yielding a mapping (Eq. (20)):

$$(\ell, w) \mapsto f \in \{0, \ldots, 99\} \tag{20}$$

This construction preserves temporal order within each label and avoids mixing early and late windows inside the same fold.

*(ii) Ten CV iterations with disjoint test blocks.* Ten iterations are executed. In iteration $k \in \{0, \ldots, 9\}$, the test set is a *contiguous block of 10 folds* per label, as shown in Eq. (21):

$$B_{\text{test}} = \{10k,\ 10k{+}1,\ \ldots,\ 10k{+}9\} \tag{21}$$

To reduce temporal leakage, immediate neighbors of the test block (when they exist) are set aside as buffers and are never used for training as shown in Eq. (22):

$$B_{\text{buf}} = \{10k{-}1,\ 10k{+}10\} \cap [0, 99]. \tag{22}$$

**Table 3**
Flow statistics per 5-s window across all processed traces.

| Window file | Label | # 5-s Windows | Min Flows | Max Flows | Mean Flows | Median Flows |
|---|---|---|---|---|---|---|
| ./Benign_0_windows.csv | BENIGN | 6667 | 12 | 227 | 58.43 | 59.0 |
| ./Benign_1_windows.csv | BENIGN | 5220 | 12 | 235 | 43.92 | 43.0 |
| ./Benign_2_windows.csv | BENIGN | 5785 | 5 | 145 | 43.07 | 43.0 |
| ./Benign_3_windows.csv | BENIGN | 2493 | 5 | 77 | 42.01 | 42.0 |
| ./DDoS-ICMP_0_windows.csv | DDoS-ICMP | 421 | 4 | 208 | 70.86 | 69.0 |
| ./DoS-TCP-0_00000_windows.csv | DoS-TCP | 111 | 2 | 37,587 | 11603.96 | 13071.0 |
| ./DoS-TCP-0_00004_windows.csv | DoS-TCP | 16 | 15751 | 70,619 | 54415.19 | 64736.5 |
| ./DoS-TCP-1_00000_windows.csv | DoS-TCP | 57 | 3 | 74,421 | 47755.04 | 63284.0 |
| ./DoS-TCP-1_00002_windows.csv | DoS-TCP | 330 | 2 | 64,624 | 1295.90 | 62.0 |
| ./DoS-TCP-2_00000_windows.csv | DoS-TCP | 48 | 4 | 65,611 | 19550.96 | 7189.0 |
| ./DoS-TCP-2_00001_windows.csv | DoS-TCP | 100 | 30 | 69,574 | 32695.59 | 34943.0 |
| ./DoS-TCP-2_00002_windows.csv | DoS-TCP | 289 | 4 | 73,915 | 14943.34 | 47.0 |
| ./DoS-TCP-2_00004_windows.csv | DoS-TCP | 35 | 3 | 76,924 | 52789.26 | 62707.0 |
| ./DDoS-TCP-0_windows.csv | DDoS-TCP | 382 | 15 | 5717 | 118.17 | 78.0 |
| ./DDoS-TCP-1_windows.csv | DDoS-TCP | 310 | 5 | 59,261 | 761.38 | 90.5 |
| ./DDoS-TCP-2_windows.csv | DDoS-TCP | 313 | 2 | 42,720 | 295.05 | 61.0 |
| ./DDoS-TCP-3_windows.csv | DDoS-TCP | 312 | 1 | 121 | 51.64 | 50.0 |
| ./Recon-OSScan_windows.csv | OSScan | 1672 | 1 | 1239 | 122.26 | 66.0 |
| ./Recon-PortScan_windows.csv | PortScan | 1522 | 2 | 1463 | 142.37 | 64.0 |

*(iii) Validation folds with adjacency blocking.* Within the remaining folds (per label), $n_{\text{val}}$=10 distinct validation folds are selected at random in each iteration (no fold is used twice within the same set), under the rule that *neither a chosen validation fold nor its immediate neighbors* enter the training set. Concretely, for each label $\ell$, the training test ($T_\ell$) is defined by Eq. (23):

$$T_\ell \;=\; \{0,\dots,99\} \setminus \left(B_{\text{test}} \cup B_{\text{buf}} \cup V_\ell \cup V_\ell^{\pm}\right), \tag{23}$$

where $V_\ell$ is the set of selected validation folds and $V_\ell^{\pm}$ are their immediate neighbors (clipped to $[0, 99]$). Three dataframes (`train`, `val`, `test`) are then created by taking, for each $\ell$, all rows whose window belongs to $T_\ell$, $V_\ell$, and $B_{\text{test}}$, respectively, and merging across labels. This construction ensures that each split contains windows from all labels while preserving temporal order and respecting adjacency buffers.

Algorithm 1 explains in pseudocode this custom CV process.

**Algorithm 1** Temporally blocked cross-validation.

For each label $\ell$: collect unique windows, sort by time, partition into 100 contiguous folds; store $(\ell, w) \mapsto f$.
**for** $b \in \{0, 10, 20, \dots, 90\}$ **do** ▷ ten disjoint test blocks
  $B_{\text{test}} \leftarrow \{b, \dots, b+9\}$, $B_{\text{buf}} \leftarrow \{b-1, b+10\} \cap [0, 99]$
  **for** each label $\ell$ **do**
    $C \leftarrow \{0, \dots, 99\} \setminus (B_{\text{test}} \cup B_{\text{buf}})$
    Choose $V_\ell \subset C$ with $|V_\ell| = 10$ (no replacement); let $V_\ell^{\pm}$ be the neighbors of $V_\ell$
    $T_\ell \leftarrow \{0, \dots, 99\} \setminus (B_{\text{test}} \cup B_{\text{buf}} \cup V_\ell \cup V_\ell^{\pm})$
  **end for**
  Build `train`, `val`, `test` by taking, for each $\ell$, all rows whose window fold is in $T_\ell$, $V_\ell$, and $B_{\text{test}}$, respectively; then merge across labels.
**end for**

### 4.3. Preliminary hyperparameter selection

Before running cross-validation, a preliminary hyperparameter selection phase was conducted to identify a stable architecture and training setup. This phase used a *single*, temporally ordered split of the data into training, validation, and test partitions, following the same *principles* as the cross-validation protocol: (i) strict temporal order (train → val → test), (ii) short adjacency buffers around validation and test boundaries excluded from training, and (iii) coverage of all labels within each partition.

In this search, only the context length $L$ was varied, since it determines the maximum number of flows retained within each window for classification. Larger $L$ values provide richer temporal and structural context but increase computational cost and latency, while too small $L$ values risk discarding relevant flows. The rest of the architectural hyperparameters were kept constant across all experiments. Table 4 summarizes the explored values of $L$ and the fixed settings for the remaining parameters.

### 4.4. Model and training setup

The classifier is a Transformer encoder operating on masked sequences of per-flow statistics *within the same window*, with `[CLS]` pooling (see Section 3). Architectural hyperparameters were fixed according to the preliminary selection summarized in Table 4:

**Table 4**
Hyperparameters explored during the preliminary search. Only $L$ was varied; the rest remained constant.

| Hyperparameter | Values explored | Fixed value |
|---|---|---|
| Context length $T$ | 2, 4, 8, 16, 32, 64, 128, 256, 512 | – |
| Model dimension $d$ | – | 64 |
| Number of attention heads $H$ | – | 4 |
| Feed-forward size $d_{ff}$ | – | 128 |
| Number of layers $N$ | – | 4 |
| Dropout | – | 0.1 |

**Table 5**
Average performance per context length $T = L + 1$ in the preliminary search. Best values per column are highlighted in bold.

| $T$ | L | F1 weighted | Balanced accuracy | Macro ROC–AUC |
|---|---|---|---|---|
| 2 | 1 | 0.113316 | 0.342437 | 0.703827 |
| 4 | 3 | 0.283512 | 0.550421 | 0.837089 |
| 8 | 7 | 0.511520 | 0.704560 | 0.912118 |
| 16 | 15 | 0.647390 | 0.728977 | 0.939112 |
| 32 | 31 | 0.898104 | 0.803363 | 0.972428 |
| 64 | 63 | 0.903683 | 0.796710 | 0.971692 |
| 128 | 127 | **0.929937** | 0.828899 | **0.988266** |
| 256 | 255 | 0.906792 | 0.816364 | 0.970001 |
| 512 | 511 | 0.923584 | **0.842887** | 0.982703 |

context length $L$ chosen from the grid search, and constant values for the rest of hyperparameters ($d$=64, $H$=4, $d_{ff}$=128, $N$=4, $dropout$=0.1). These settings remained unchanged across all CV folds.

Training used cross-entropy with Class Weighting to mitigate imbalance, Gradient Clipping, and Early Stopping driven by validation loss. In each iteration, a feature scaler was fit on the `train` windows (valid tokens only) and reused unchanged for `val/test` windows; specifically, a StandardScaler was applied. At inference time, the classifier head consumed the final `[CLS]` embedding to produce one score per class, which was then normalized into probabilities via a Softmax.

### 4.5. Metrics and reporting

For each held-out test block, the following metrics are reported: (i) Balanced Accuracy, (ii) weighted Harmonic mean of precision and recall ($F_1$-score), and (iii) Macro ROC-AUC in a One-vs-Rest multi-class evaluation scheme (OVR) setting, together with per-class ROC curves. Confusion Matrix and complete classification reports (precision, recall, $F_1$-score per class) are also produced. All metrics are computed on window-level predictions from the Transformer.

### 4.6. Hardware setup

The experimental evaluation was performed on a dedicated workstation comprising an AMD Ryzen 7500F CPU, 32 GB of DDR5 RAM, and an NVIDIA RTX 4060 Ti GPU with 16 GB of VRAM, which was used to accelerate model training and inference.

## 5. Results

This section presents the experimental results obtained from the proposed methodology. The analysis begins with an examination of the influence of the context length parameter on model performance, identifying the optimal trade-off between predictive accuracy and computational efficiency. Subsequently, the outcomes of a 10-fold temporally blocked cross-validation procedure are reported to assess the model's robustness and generalization under realistic, time-dependent conditions. Detailed analyses are then provided regarding the distribution of performance metrics, error structure, and class separation margins. Additionally, the real-time feasibility of the proposed approach is evaluated, with a focus on the computational cost and latency trade-offs when applied in practical IoT network intrusion detection scenarios. Collectively, these results offer a comprehensive evaluation of the proposed approach in terms of both predictive reliability and operational practicality.

### 5.1. Hyperparameter search for context length

Table 5 summarizes the effect of the *context length* $L$ on the average performance over a temporally ordered split (train→val→test) with adjacency buffers. As $L$ grows, the encoder can incorporate more flows per window, enriching the representation of IoT activity. However, larger $L$ also implies higher memory consumption and inference latency.

Fig. 5 illustrates the evolution of the three main metrics: weighted $F_1$, balanced accuracy, and macro ROC–AUC as a function of the effective number of flows retained per window ($L = T-1$). The curves reveal a clear "elbow" around $L$=127: performance improves

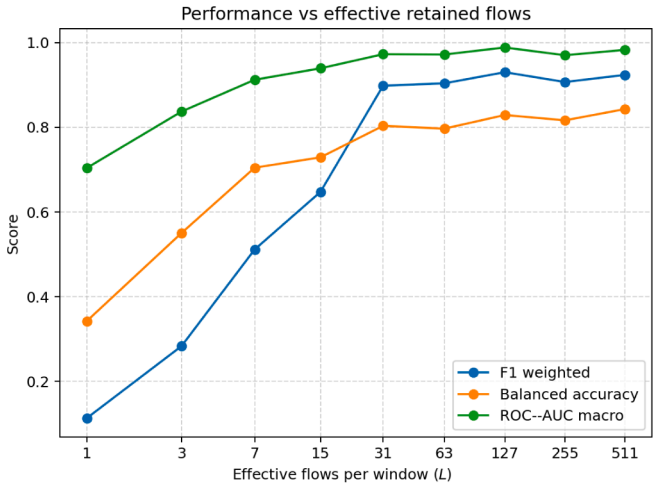


**Fig. 5.** Performance metrics versus effective retained flows ($L$). The elbow around $L$=127 indicates the point of diminishing returns.

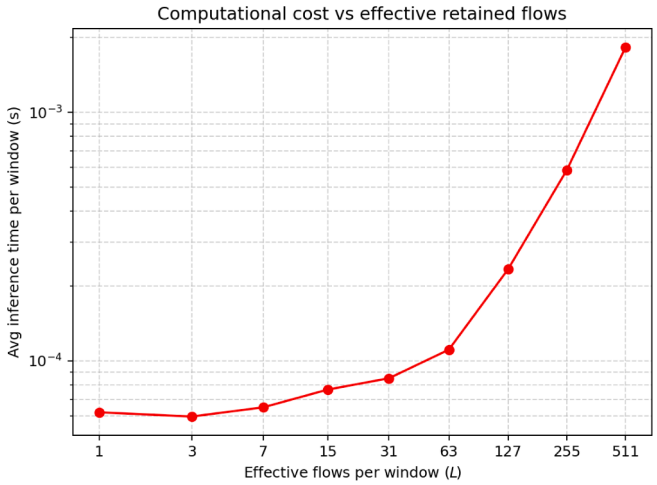


**Fig. 6.** Average inference time per window versus effective retained flows ($L$). Computational cost increases sharply with larger windows, limiting the practicality of very large $L$.

steeply up to this point, but further increases provide only marginal gains. For instance, the best weighted $F_1$ and macro ROC–AUC are both attained at $L$=127 (0.9299 and 0.9883, respectively), while balanced accuracy continues to increase slightly up to $L$=511 (0.8429). Beyond $L$=127, the performance gains are insignificant when compared to the additional computational cost.

To quantify the efficiency trade-off, Fig. 6 reports the average inference time per window as a function of $L$. The cost remains almost constant for small windows, grows moderately up to $L$=63, and then increases sharply once the context exceeds $L$=127. This accelerated, exponential-like growth makes very large windows impractical: while $L$=511 yields a slight gain in balanced accuracy, the corresponding inference time becomes disproportionately high.

Considering both predictive performance and efficiency, the context length was fixed $L$=127 for all subsequent cross-validation experiments. This value provides sufficient temporal context to maximize weighted $F_1$ and macro ROC–AUC while avoiding the unnecessary computational overhead of very large windows.

### 5.2. *10-fold temporally blocked cross-validation*

To obtain temporally faithful estimates while covering all classes, 10-fold temporally blocked cross-validation with adjacency buffers was executed (Section 4.2). The individual confusion matrices for each fold can be found in Fig. A.1 (Appendix A).

#### 5.2.1. *Summary statistics*

Table 6 reports summary statistics across the ten temporally ordered test blocks. On average, the model sustains very high performance with limited variability across folds.

**Table 6**
10-fold temporally blocked cross-validation: mean, standard deviation (std), min, max, median and IQR (Q1–Q3) across test blocks.

| Metric | Mean | Std | Min | Max | Median | Q1 | Q3 | IQR |
|---|---|---|---|---|---|---|---|---|
| F1 weighted | 0.9788 | 0.0228 | 0.9187 | 0.9950 | 0.9861 | 0.9785 | 0.9914 | 0.0129 |
| Balanced accuracy | 0.9535 | 0.0288 | 0.8844 | 0.9862 | 0.9583 | 0.9450 | 0.9730 | 0.0280 |
| Macro ROC–AUC | 0.9959 | 0.0045 | 0.9850 | 0.9999 | 0.9979 | 0.9939 | 0.9983 | 0.0043 |

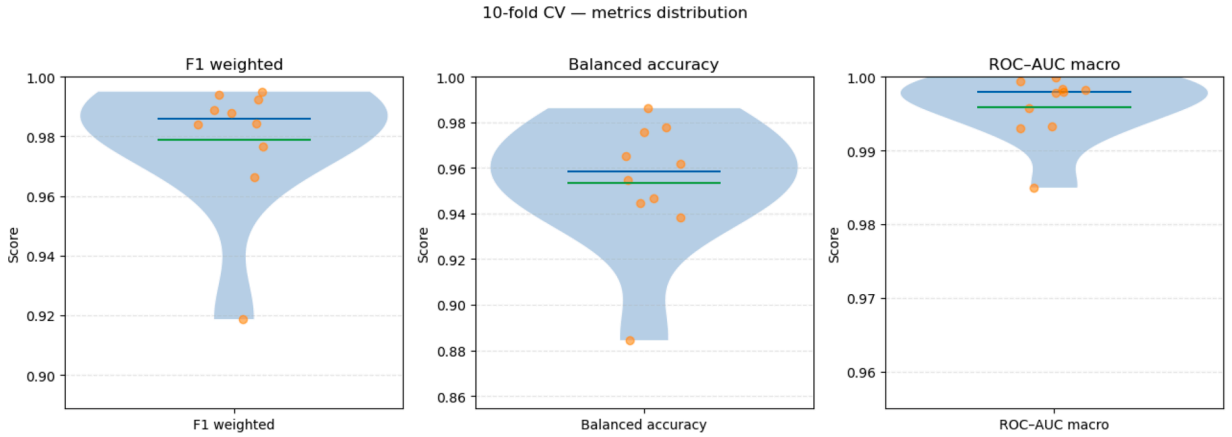


**Fig. 7.** Metrics distribution of the 10-fold temporally blocked cross-validation. Dots are per-fold scores; horizontal lines indicate mean (green color) and median (blue color). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

On average across folds ($n$=10), the model achieves $F_1$ (weighted) = 0.9788 ± 0.0141, balanced accuracy = 0.9535 ± 0.0179, and ROC–AUC (macro) = 0.9959 ± 0.0028, reported as mean ± 95% $t$-based confidence intervals (i.e., $\bar{x} \pm t_{0.025,9}\, s/\sqrt{10}$).
The violin plots in Fig. 7 show compact distributions for all metrics. One of the ten temporal blocks performs lower than the rest (weighted $F_1$ = 0.9187, BA = 0.8844), while the remaining nine cluster near the medians (0.9861 and 0.9583; Table 6). Even in that block, absolute performance remains strong, and the Interquartile Ranges (IQR) are small (IQR = 0.0129 for weighted $F_1$ and 0.0280 for BA), indicating that the drop is fold-specific rather than systematic.

### 5.2.2. Error structure

Fig. 8 shows a row-normalized confusion matrix with a strong diagonal, indicating high per-class recall: Benign 98.5% (19,865), DDoS-ICMP 92.6% (390), DDoS-TCP 96.1% (1,263), DoS-TCP 93.1% (916), OSScan 96.6% (1,615), and PortScan 93.5% (1,449). In intrusion-detection terms, *recall or true positive rate on attack classes* is the primary objective (missed attacks are costlier than extra alerts), while *low false-positive rate on Benign* avoids operator overload.

Benign windows are rarely mislabelled (overall 1.5%): the largest spillover is to PortScan (0.9%; 185), followed by DDoS-TCP (0.3%; 52), OSScan (0.2%; 35) and DDoS-ICMP (0.1%; 16), indicating a low false-alert load on normal traffic.
Confusions concentrate in semantically adjacent pairs:

(i) *DoS vs. DDoS (TCP)* – 2.3% of DoS-TCP is predicted as DDoS-TCP (23), whereas the reverse is 0.8% (11).
(ii) *Scanning variants* – PortScan is misclassified as OSScan at 2.0% (31), and OSScan is misclassified as PortScan at 1.4% (23).
(iii) *DDoS-ICMP leakage* – 3.8% to DDoS-TCP (16) and 2.6% to Benign (11), making DDoS-ICMP the relatively hardest class (92.6% recall).

These misclassifications are not random but follow semantic proximity between classes. DoS and DDoS (TCP) share signatures and differ mainly in the number of flows; since the model caps the context at `T = 128`, it cannot always capture extreme intensity differences, making this boundary more difficult. PortScan and OSScan both represent scanning behaviors with minor operational differences, explaining their overlap. Finally, DDoS-ICMP is the hardest class because its patterns resemble both benign ICMP and high-rate DDoS, which blurs the separation margin.

### 5.2.3. Separation margins

Fig. 9 shows wide margins for most classes: Benign, OSScan and PortScan with Area Under the Curve (AUC) ≈ 0.999, DDoS-TCP ≈ 0.995, and narrower margins for DoS-TCP (≈ 0.972) and DDoS-ICMP (≈ 0.964). At False Positive Rate (FPR) ∼1%, True Positive Rate (TPR) remains ≥ 0.99 for Benign, OSScan, PortScan and DDoS-TCP; in contrast, DDoS-ICMP drops earlier, consistent with its lower recall and confusions towards Benign and DDoS-TCP (Fig. 8). DoS-TCP shows intermediate overlap with DDoS-TCP when the number of active flows exceeds the context limit ($L$ = 127).

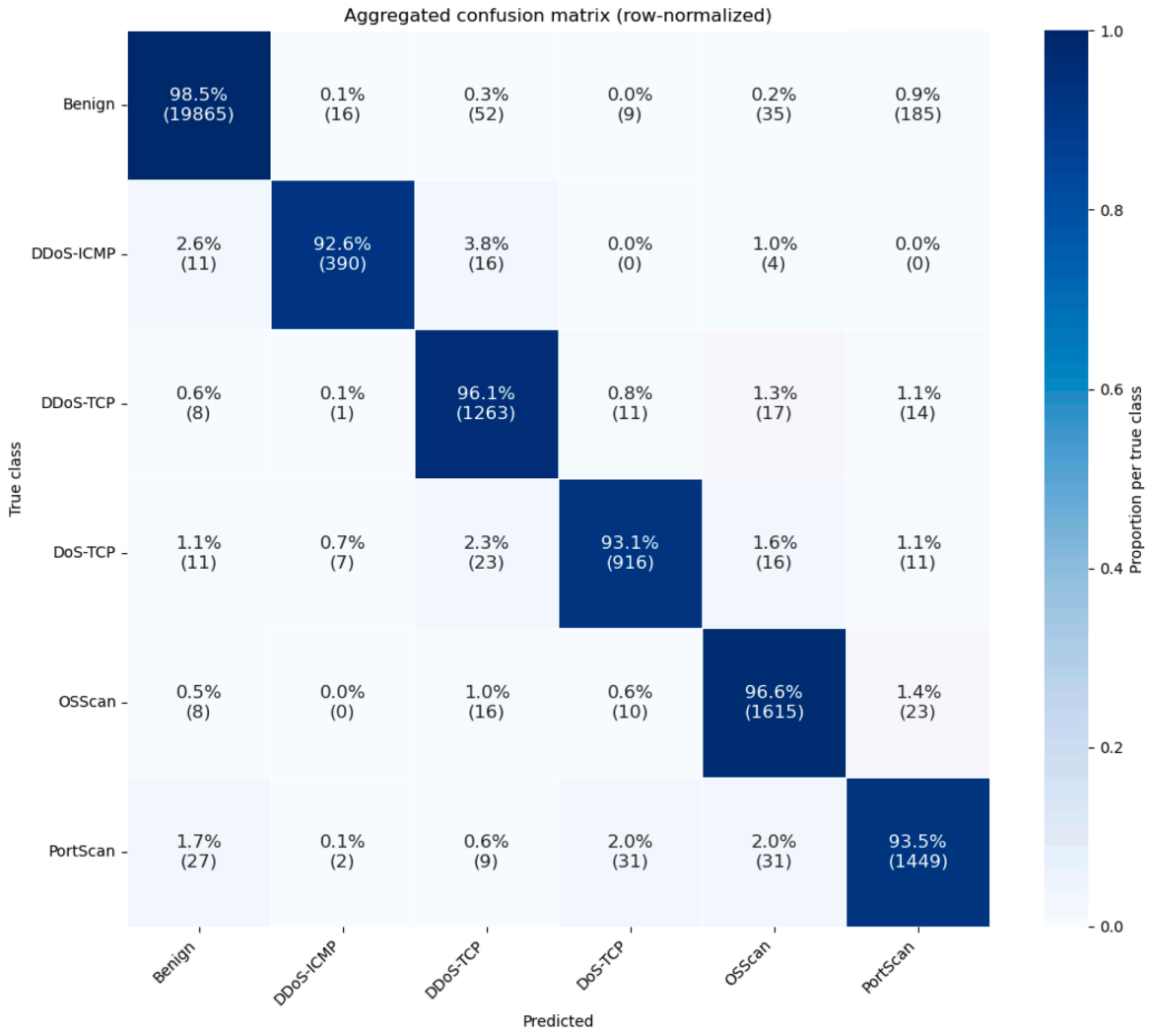


**Fig. 8.** Aggregated confusion matrix across the 10-fold temporally blocked cross-validation.

**Table 7**
Average end-to-end latency per 5 s window by traffic type. The worst case is in bold.

| Traffic type | Flow extraction (s/w) | Windowing (s/w) | Total (s/w) |
|---|---|---|---|
| Benign | 0.0302 | 0.0015 | 0.0317 |
| DDoS–ICMP | 0.4828 | 0.0017 | 0.4845 |
| **DDoS–TCP** | **2.2042** | **0.1285** | **2.3327** |
| DoS–TCP | 1.0826 | 0.0689 | 1.1515 |
| OSScan | 0.0453 | 0.0022 | 0.0475 |
| PortScan | 0.0453 | 0.0026 | 0.0479 |
| Transformer inference (all) | 0.0003 (≈0.3 ms) | | – |

### 5.3. Real-time feasibility assessment

To evaluate operational viability, one representative 200-s PCAP trace (40 windows of 5 s) was selected per traffic type from the CICIoT2023 dataset. For each case, the latency of the three processing stages–(i) flow extraction using CICFlowMeter, (ii) windowing and tensorization with the preprocessing script, and (iii) Transformer inference–was measured separately. The inference cost was derived from the curve in Fig. 6, where the point at $L = 127$ corresponds to an average inference time of 0.3 ms per window ($3 \times 10^{-4}$ s). The latency values presented in Table 7 are expressed as seconds per 5-s window of traffic processed (s/w), with each stage considered separately.

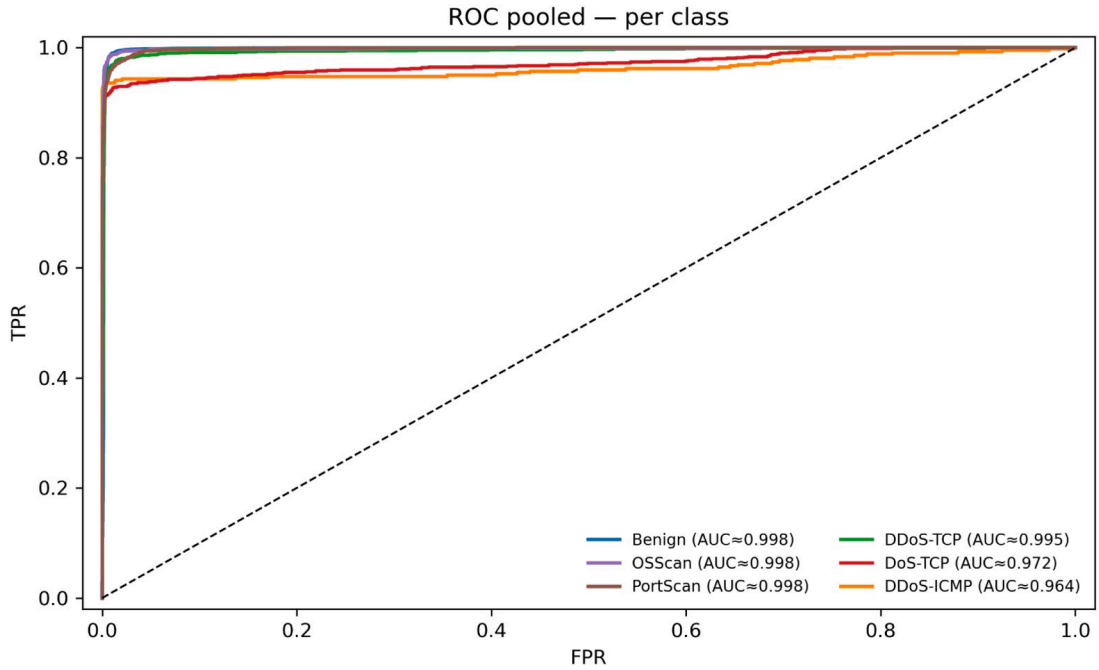


**Fig. 9.** Pooled ROC curves across the 10-fold temporally blocked cross-validation.

The total latency per window can be expressed as:

$$t_{total} = t_{conv} + t_{win} + t_{inf} \approx 2.3330\,\text{s} \text{ (worst-case)}.$$

Although the worst-case latency (2.33 s) represents nearly half of the 5-s window duration, each prediction still corresponds to a 5-s traffic segment. Therefore, even under the heaviest DDoS–TCP conditions, the system processes traffic faster than it is generated, satisfying real-time operational requirements.
The components of the total latency are broken down as follows:

- $t_{conv}$ (flow conversion time): The time taken to convert raw network traffic into a flow-based format using CICFlowMeter. This step involves extracting the necessary flow features from the raw packet captures (PCAPs) to prepare them for input into the model.
- $t_{win}$ (windowing and tensorization time): The time required for aggregating the network flows into fixed-duration windows (5 s), followed by the tensorization process that formats the data into a model-ready structure. This is performed by the preprocessing script, which ensures that the input data is correctly aligned with the model's requirements.
- $t_{inf}$ (Transformer inference time): The inference time of the Transformer model for a single 5-s window. This step involves using the self-attention mechanism of the Transformer to capture dependencies between the concurrent flows and predict the class of each individual window. This stage is almost instantaneous, taking approximately 0.3 ms per window.

## 6. Discussion

The experimental results confirm that the proposed flow-to-window Transformer achieves competitive intrusion detection performance in IoT environments while preserving temporal consistency through temporally blocked cross-validation. Averaged over the ten temporally ordered folds, the model maintains a weighted $F_1$-score of 0.9788 and a macro ROC–AUC of 0.9959, with standard deviations of 0.0228 and 0.0045 respectively, demonstrating both predictive strength and stability across time. Balanced accuracy also remains high (mean 0.9535, SD 0.0288), indicating that the model performs uniformly well across classes.

A detailed inspection of the fold-wise results and confusion matrices reveals that the performance degradation in the first temporal block (00–09) originates mainly from increased false positives between Benign and PortScan attacks. In this block, 132 benign windows are misclassified as PortScan, whereas only minor confusion is observed between attack classes, such as DoS and DDoS. This suggests that the earliest traffic captures include benign connection patterns–short or exploratory flows toward multiple destinations–that statistically resemble scan behavior when aggregated into 5-s windows. Subsequent folds exhibit significantly cleaner diagonals, indicating that the model rapidly stabilizes once the benign traffic distribution becomes consistent over time.

This pattern aligns with a mild *covariate shift* over time: the statistical distribution of flows in the earliest PCAP segments differs from that of later intervals. Early windows show heterogeneous traffic densities, with some containing only a few flows and others containing thousands of flows. Because the Transformer operates with a fixed context length ($L = 127$), extreme sparsity or burstiness may truncate contextual information or dilute the relative intensity of active flows, making certain benign or low-intensity patterns

appear anomalous. Consequently, the lower performance in the first block arises from temporal shifts in the traffic composition and flow density rather than from structural weaknesses in the model itself.

Class-wise statistics support this interpretation. PortScan (mean F1 = 0.916, SD = 0.094) and DoS–TCP (mean F1 = 0.944, SD = 0.076) exhibited the largest temporal dispersion, with minima of 0.6667 and 0.7425, respectively, in the 00–09 block. DDoS–ICMP also fluctuates moderately (mean F1 = 0.932, SD = 0.073), reaching 0.8205 in 30–39. The remaining classes, including benign traffic, maintain stable F1 values above 0.98 across all folds. The residual confusions therefore correspond to semantically adjacent categories whose aggregated flow statistics (duration, packet and byte counts, destination fan-out) naturally overlap, confirming that errors arise from intrinsic feature similarity rather than model bias.

Regarding the number of flows fed into the model, our empirical analysis shows that expanding the contextual window to include more concurrent flows leads to a measurable improvement in performance compared to reduced-context configurations that approximate traditional per-flow approaches. These gains increase steeply as the number of retained flows increases, up to $L = 127$, beyond which improvements become marginal, illustrating a natural trade-off between contextual richness and inference latency. The Transformer encoder efficiently captures intra-flow and inter-flow dependencies via self-attention without explicit recurrence, enabling parallel computation and superior scalability for large IoT deployments.

Notably, the proposed approach operates on a different atomic unit of inference than most existing intrusion detection methods. While the majority of prior work performs classification at the level of individual network flows, our model produces predictions at the level of fixed-duration temporal windows containing multiple concurrent flows. As a result, a direct one-to-one experimental comparison with per-flow classifiers under a shared evaluation protocol would not be methodologically appropriate, since predictions are generated over different units and temporal contexts.

Unlike prior CNN–RNN hybrid approaches [19,24,25], which operate at the level of individual network flows and rely on sequential recurrence, the proposed architecture explicitly models relationships among multiple concurrent flows within a shared temporal window. By treating flows as the atomic units of the input sequence and leveraging self-attention, the model captures inter-flow context while enabling fully parallel inference without recurrent processing.

In contrast to other Transformer-based approaches [36,38,40–43], the use of self-attention is typically confined to representations derived from individual network flows or aggregated window-level embeddings, preserving a per-flow or per-window prediction granularity. As a result, the capacity of Transformers to model interactions among concurrent communications is not fully exploited in current intrusion detection settings. This limitation is exemplified by the approach of Kozik et al. [40], where self-attention models temporal dependencies across sequences of aggregated time-window embeddings rather than interactions among concurrent flows within a single window.

The proposed method directly addresses this limitation by shifting the unit of analysis from isolated flows or aggregated windows to short temporal windows represented as sets of concurrent flows. In this formulation, self-attention operates directly over complete flow representations coexisting within the same window, enabling explicit modeling of inter-flow contextual relationships.

*Real-time feasibility.* To evaluate real-time viability, the latency of each stage in the end-to-end processing pipeline was measured using representative 200-s PCAP traces (40 windows of 5 s each) from different attack types of the CICIoT2023 dataset. For each traffic category, the following metrics were recorded: (i) the flow extraction time per window using CICFlowMeter, (ii) the time required by our windowing and tensorization script, and (iii) the Transformer inference latency, derived from the curve in Fig. 6, which shows the computational cost as a function of the number of effective retained flows. The point at $L = 127$ corresponds to an average inference time of $0.3\,\mathrm{ms}$ per window ($3 \times 10^{-4}$ s).

The empirical latency analysis presented in Section 5.3 demonstrates that the overall pipeline operates within real-time constraints. Even in the worst-case DDoS–TCP scenario, the end-to-end latency per 5-s window (2.33 s) remains below the window duration, meaning the system can process traffic faster than it is generated. The Transformer inference itself is negligible (0.3 ms per window), while flow extraction dominates total delay. Therefore, real-time operation is feasible for most IoT traffic types, and parallelized flow extraction could further enhance the processing margin under heavy-load conditions.

*From Lab to IoT deployment considerations.* While the aforementioned latency analysis confirms real-time feasibility in a controlled environment, deploying the proposed model in a real IoT network presents distinct challenges. The primary bottleneck is not the model's inference speed, which is minimal, but the resource-intensive flow extraction stage. In a practical setting, this module would need to be integrated directly into the network data path, such as within a firewall, router, or dedicated edge security appliance. Crucially, the performance metrics reported in this study were obtained on a system with a high-end GPU (NVIDIA RTX 4060 Ti) and substantial memory. Achieving comparable latency on typical, resource-constrained IoT gateway hardware would require significant optimization, including model quantization, pruning, and the use of efficient feature extractors. Furthermore, real deployment must handle variable, non-synthetic traffic, necessitating robust continuous learning and adaptation mechanisms to maintain detection accuracy beyond the curated training dataset. These topics should be addressed in follow-up research works.

Despite these strengths, several limitations and directions for future work remain; for clarity and to avoid extending the Discussion further, they are summarized in Section 7.

## 7. Conclusion

This work introduced a Transformer-based intrusion detection framework for IoT networks that aggregates network flows into fixed-duration windows and treats each flow as a token within the input sequence. This flow-to-window formulation enables the

model to learn contextual relationships across concurrent flows through self-attention, bridging the gap between per-flow and fully sequential methods and aligning the learning process with the operational granularity of real intrusion detection systems.

Across ten temporally ordered folds, the model achieved a weighted F1-score of 97.9% and a macro ROC–AUC of 99.6%, with minimal variance over time, demonstrating both accuracy and temporal robustness. The detailed temporal and class-wise analysis revealed that most misclassifications occurred in the earliest time block (00–09), mainly involving benign traffic mislabeled as PortScan. This behavior was attributed to distributional shifts in the initial traffic captures rather than model bias, confirming the overall stability of the proposed architecture across time.

Beyond predictive performance, an empirical latency evaluation was conducted to assess the real-time feasibility of the complete processing pipeline. The results showed that, under the chosen configuration ($L = 127$), Transformer inference is virtually instantaneous (0.3 ms per 5-s window), and the total end-to-end latency–including flow extraction and windowing–remains below the window duration even in the heaviest DDoS–TCP scenarios (2.33 s per 5 s of traffic). Therefore, the system is capable of maintaining real-time throughput, processing network traffic faster than it is generated.

Despite these strengths, several limitations remain and motivate future research. First, the evaluation covered only a subset of attack scenarios from the CICIoT2023 dataset; extending the analysis to all 33 scenarios, as well as to additional public datasets such as ToN-IoT or BoT-IoT, would further validate the generalization capability of the proposed approach. Second, although fixed 5-s temporal windows provide a practical balance between temporal resolution and contextual richness, adaptive or overlapping windowing strategies could improve responsiveness to highly bursty or transient traffic patterns. In addition, incorporating protocol metadata or device-level telemetry [41] may help disambiguate semantically similar traffic behaviors and improve robustness in heterogeneous IoT environments. Finally, while the impact of the maximum number of retained flows per window ($L$) was empirically analyzed in this study, a systematic sensitivity analysis of other design parameters in the processing pipeline–such as the idle and activity timeouts used during flow extraction and the temporal window size–remains beyond the scope of the present work. The values adopted here represent a practical compromise between near real-time responsiveness and statistical representativeness, and will be the subject of dedicated future investigations. In addition, further research is needed to improve the interpretability of the attention mechanism, particularly to better understand how attention weights reflect and prioritize relevant inter-flow interactions in IoT traffic.

## Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work, the authors used generative AI-based tools to assist with language refinement and formatting suggestions. After using these tools, the authors reviewed and edited all content as needed and take full responsibility for the final version of the manuscript.

## CRediT authorship contribution statement

**Sergio Martin-Reizabal:** Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Adrian Caballero-Quiroga:** Writing – review & editing, Investigation; **Beatriz Gil-Arroyo:** Writing – review & editing, Investigation; **Nuño Basurto:** Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Conceptualization; **Ruben Ruiz-Gonzalez:** Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Conceptualization.

## Data availability

The data supporting the findings of this study were derived from the public CICIoT2023 dataset; however, the processed flow-to-window aggregated files and the trained Transformer model were not made publicly available due to institutional storage and licensing restrictions. They can be obtained from the corresponding author upon reasonable request.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Appendix A. Fold-wise confusion matrices

### Glossary

*Terms*

**Activity Timeout** Upper bound on the active duration allowed for a flow; if a flow remains active longer than this limit, it is forcefully terminated and a new flow must be started.

**Attention Mechanism** Neural operation that allows a model to focus on the most relevant parts of the input when making predictions.

**Balanced Accuracy Metric** computed as the average of per-class recalls, more robust under class imbalance than raw accuracy.

**CICFlowMeter** Tool that converts packet traces into bidirectional flows and computes per-flow statistics used for traffic analysis.

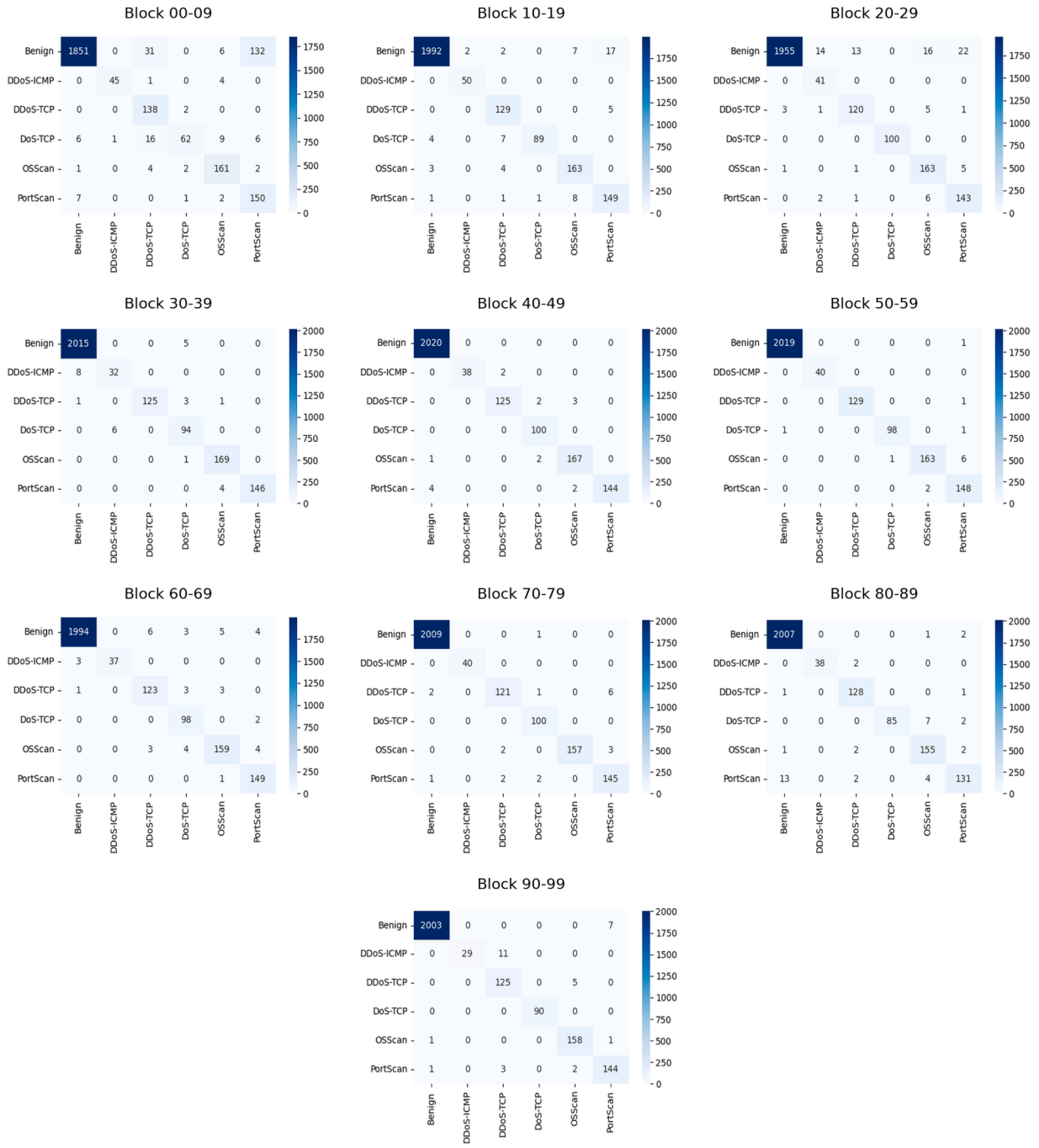


**Fig. A.1.** Individual confusion matrices for each block under 10-fold temporally blocked cross-validation.

**Class Weighting** Loss reweighting strategy that increases/decreases each class contribution to mitigate label imbalance.

**Confusion Matrix** Table that summarizes counts of true vs. predicted labels for each class.

**Early Stopping** Training regularization that halts when validation performance stops improving to prevent overfitting. **Edge Computing** Distributed computing paradigm where data is processed near the source (e.g., gateways or routers) to reduce latency and resource usage.

**FIN Signal** TCP control flag indicating that a host has finished sending data on a connection.

**Five-Tuple** Canonical flow identifier: source IP, destination IP, source port, destination port, and transport protocol.

**Flow Timeout** Idle timeout after which a flow is closed if no packets are observed for its five-tuple.

**Flow-to-Window Aggregation** Temporal aggregation method where multiple network flows within the same time window are grouped together to form the model's input sequence.

**Gradient Clipping** Technique that caps gradient norm or value during backpropagation to stabilize training.

**Intrusion Detection** Process of monitoring network traffic to identify malicious or abnormal activities.

**Machine Learning** Field of artificial intelligence that enables systems to learn patterns from data and make predictions or decisions without being explicitly programmed.

**Macro ROC-AUC** Evaluation metric that computes the Area Under the Receiver Operating Characteristic curve (ROC-AUC) independently for each class and averages them, giving equal weight to all classes regardless of their frequency.

**Mask** Binary indicator used to mark valid tokens versus padding positions so attention and losses ignore padded elements.

**MLP** Multi-Layer Perceptron - feed-forward neural network consisting of multiple fully connected layers.

**Multi-Head Attention** Attention mechanism that projects queries, keys, and values into multiple subspaces (heads) to capture diverse interactions in parallel.

**OSScan** Operating system scan reconnaissance technique that probes stack/network fingerprints to infer the target OS.

**PortScan** Reconnaissance technique that probes ranges of ports to discover open services on a host or subnet.

**Recurrent Model** Type of neural network architecture that processes sequential data by retaining information about previous inputs through recurrent connections.

**Reinforcement learning** paradigm in which an agent learns to take actions that maximize a cumulative reward through interaction with an environment.

**Self-Attention** Mechanism used in Transformers that computes pairwise relationships among tokens in the same sequence to capture contextual dependencies.

**Softmax** Function that converts logits into a probability distribution over classes.

**StandardScaler** Feature normalization that transforms each feature to zero mean and unit variance, typically estimated on the training set.

**Temporal Dependency** Relationship between events or data points over time, essential for sequential modeling in network traffic.

**Temporally blocked cross-validation** Cross-validation strategy for time-dependent data in which samples are split into contiguous, time-ordered blocks. In the context of flow-to-window aggregation, this prevents temporally adjacent windows (potentially sharing the same network flows) from being assigned to different partitions, thereby avoiding data leakage between training, validation, and test sets. The full protocol is described in Section 4.2.

**Tensorization** Transformation that maps variable-length, structured records (e.g., flows within a window) into fixed-shape tensors suitable for batch processing.

**token** Elementary input element of the Transformer architecture. In this work, tokens represent individual network flows and are used as internal representations to model intra- and inter-flow relationships within a fixed-duration temporal window, which constitutes the atomic unit of inference.

**Transformer** Neural architecture based on self-attention that processes input sequences in parallel, avoiding recurrence.

**WindowDataset** Dataset abstraction that yields, for each time window, the scaled feature matrix, the validity mask, and the window label.

*Acronyms*

***k*-NN** *k*-Nearest Neighbors.

**AUC** Area Under the Curve.

**BiLSTM** Bidirectional Long Short-Term Memory.

**BoT-IoT** BoT-IoT dataset for IoT botnet traffic.

**BPE** Byte-Pair Encoding.

**CIC-IDS2017** Canadian Institute for Cybersecurity IDS 2017 dataset.

**CICIoT2023** Canadian Institute for Cybersecurity IoT 2023 dataset.

**CNN** Convolutional Neural Network.

**CSV** Comma-Separated Values.

**CV** Cross-Validation.

**DDoS** Distributed Denial of Service.

**DL** Deep Learning.

**DQ-IDS** Deep Q-learning Intrusion Detection System.

**$F_1$-score** Harmonic mean of precision and recall.

**FFN** Feed-Forward Network.

**FL** Federated Learning.

**FPR** False Positive Rate.

**GRU** Gated Recurrent Unit.

**IDS** Intrusion Detection System.

**IoT** Internet of Things.

**IoT-23** IoT-23 dataset.

**IoTID20** IoTID20 dataset for IoT intrusion detection.

**IQR** Interquartile Ranges.

**KD** Knowledge Distillation.

**LSTM** Long Short-Term Memory.
**MitM** Man-in-the-Middle.
**MQTT** Message Queuing Telemetry Transport.
**MQTTset** MQTTset dataset.
**N-BaIoT** N-BaIoT dataset.
**NIDS** Network Intrusion Detection System.
**NSL-KDD** NSL-KDD dataset.
**OSNN** Optimized Sequential Neural Network.
**OVR** One-vs-Rest multi-class evaluation scheme.
**PCA** Principal Component Analysis.
**PCAP** Packet CAPture.
**RNN** Recurrent Neural Network.
**ROC-AUC** Area Under the Receiver Operating Characteristic curve.
**SVM** Support Vector Machine.
**TCP** Transmission Control Protocol.
**ToN-IoT** Telemetry of Networked IoT dataset.
**TPR** True Positive Rate.
**UDP** User Datagram Protocol.
**UNSW-NB15** UNSW-NB15 dataset.
**X-IIoTID** X-IIoTID dataset.
**XAI** Explainable Artificial Intelligence.

## References


[1] E. Dritsas, M. Trigka, A survey on cybersecurity in IoT, Future Internet 17 (1) (2025). https://doi.org/10.3390/FI17010030
[2] Statista, IoT connections worldwide 2034 | Statista, 2025, https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.
[3] E. Gyamfi, A. Jurcut, Intrusion detection in internet of things systems: a review on design approaches leveraging multi-access edge computing, machine learning, and datasets, Sensors 22 (10) (2022). https://doi.org/10.3390/S22103744
[4] B.R. Kikissagbe, M. Adda, Machine learning-based intrusion detection methods in IoT systems: a comprehensive review, Electronics 13 (18) (2024). https://doi.org/10.3390/electronics13183601
[5] Y. Zhang, R.C. Muniyandi, F. Qamar, A review of deep learning applications in intrusion detection systems: overcoming challenges in spatiotemporal feature extraction and data imbalance, Appl. Sci. 15 (3) (2025). https://doi.org/10.3390/app15031552
[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 6000–6010.
[7] U.C. Akuthota, L. Bhargava, Transformer-based intrusion detection for IoT networks, IEEE Internet Things J. 12 (5) (2025). https://doi.org/10.1109/JIOT.2025.3525494
[8] Z. Long, H. Yan, G. Shen, X. Zhang, H. He, L. Cheng, A transformer-based network intrusion detection approach for cloud security, J. Cloud Comput. 13 (1) (2024). https://doi.org/10.1186/s13677-023-00574-9
[9] E.C.P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, A.A. Ghorbani, CICIoT2023: a real-time dataset and benchmark for large-scale attacks in IoT environment, Sensors 23 (13) (2023). https://doi.org/10.3390/S23135941
[10] E. Schiller, A. Aidoo, J. Fuhrer, J. Stahl, M. Ziörjen, B. Stiller, Landscape of IoT security, Comput. Sci. Rev. 44 (2022) 100467. https://doi.org/10.1016/j.cosrev.2022.100467
[11] S.K. Sharma, B. Bhushan, N.C. Debnath (Eds.), Security and Privacy Issues in IoT Devices and Sensor Networks, Elsevier, 2021.
[12] H. Zhang, Development of an intelligent intrusion detection system for IoT networks using deep learning, Discover Internet Things 5 (1) (2025). https://doi.org/10.1007/s43926-025-00177-7
[13] A. Sharma, K. Bhushan, A comprehensive survey on IoT security: challenges, security issues, and countermeasures, Comput. Sci. Rev. 59 (2026) 100839. https://doi.org/10.1016/j.cosrev.2025.100839
[14] B.B. Zarpelão, R.S. Miani, C.T. Kawakani, S.C. de Alvarenga, A survey of intrusion detection in internet of things, J. Netw. Comput. Appl. 84 (2017) 25–37. https://doi.org/10.1016/j.jnca.2017.02.009
[15] H.C. Altunay, Z. Albayrak, A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks, Eng. Sci. Technol. Int. J. 38 (2023) 101322. https://doi.org/10.1016/j.jestch.2022.101322
[16] S. Wang, W. Xu, Y. Liu, Res-TranBiLSTM: an intelligent approach for intrusion detection in the internet of things, Comput. Netw. 235 (2023) 109982. https://doi.org/10.1016/j.comnet.2023.109982
[17] A. Nazir, J. He, N. Zhu, S.S. Qureshi, S.U. Qureshi, F. Ullah, A. Wajahat, M.S. Pathan, A deep learning-based novel hybrid CNN-LSTM architecture for efficient detection of threats in the IoT ecosystem, Ain Shams Eng. J. 15 (7) (2024) 102777. https://doi.org/10.1016/j.asej.2024.102777
[18] F.S. Alsubaei, Smart deep learning model for enhanced IoT intrusion detection, Sci. Rep. 15 (1) (2025). https://doi.org/10.1038/s41598-025-06363-5
[19] K.O. Adefemi, M.B. Mutanga, O.A. Alimi, A hybrid CNN–GRU deep learning model for IoT network intrusion detection, J. Sensor Actuator Netw. 14 (5) (2025). https://doi.org/10.3390/jsan14050096
[20] I. Ullah, Q.H. Mahmoud, A scheme for generating a dataset for anomalous activity detection in IoT networks, in: C. Goutte, X. Zhu (Eds.), Advances in Artificial Intelligence, Springer International Publishing, Cham, 2020, pp. 508–520. https://doi.org/10.1007/978-3-030-47358-7_52
[21] N. Koroniotis, N. Moustafa, E. Sitnikova, B. Turnbull, Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-IoT dataset, Future Gener. Comput. Syst. 100 (2019) 779–796. https://doi.org/10.1016/j.future.2019.05.041
[22] N.W. Khan, M.S. Alshehri, M.A. Khan, S. Almakdi, N. Moradpoor, A. Alazeb, S. Ullah, N. Naz, J. Ahmad, A hybrid deep learning-based intrusion detection system for IoT networks, Math. Biosci. Eng. 20 (8) (2023) 13491–13520. https://doi.org/10.3934/mbe.2023602
[23] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, A. Anwar, TON_IoT telemetry dataset: a new generation dataset of IoT and IIoT for data-driven intrusion detection systems, IEEE Access 8 (2020) 165130–165150. https://doi.org/10.1109/ACCESS.2020.3022862
[24] T. Sasi, A.H. Lashkari, R. Lu, P. Xiong, S. Iqbal, An efficient self attention-based 1D-CNN-LSTM network for IoT attack detection and identification using network traffic, J. Inform. Intell. (2024). https://doi.org/10.1016/j.jiixd.2024.09.001
[25] A.M. Alashjaee, Deep learning for network security: an attention-CNN-LSTM model for accurate intrusion detection, Sci. Rep. 15 (2025) 21856. https://doi.org/10.1038/s41598-025-07706-y
[26] M. Tavallaee, E. Bagheri, W. Lu, A.A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1–6. https://doi.org/10.1109/CISDA.2009.5356528

[27] M.A. Hossain, Deep Q-learning intrusion detection system (DQ-IDS): a novel reinforcement learning approach for adaptive and self-learning cybersecurity, ICT Express 11 (5) (2025) 875–880. https://doi.org/10.1016/j.icte.2025.05.007
[28] M.A. Hossain, Deep learning-based intrusion detection for IoT networks: a scalable and efficient approach, EURASIP J. Inform. Secur. 2025 (1) (2025) 28. https://doi.org/10.1186/s13635-025-00202-w
[29] M.A. Ali, S.A.H. Al-Sharafi, Intrusion detection in IoT networks using machine learning and deep learning approaches for MitM attack mitigation, Discover Internet Things 5 (2025) 48. https://doi.org/10.1007/s43926-025-00104-w
[30] B. Olanrewaju-George, B. Pranggono, Federated learning-based intrusion detection system for the internet of things using unsupervised and supervised deep learning models, Cyber Secur. Appl. 3 (2025) 100068. https://doi.org/10.1016/j.csa.2024.100068
[31] V.T. Nguyen, R. Beuran, FedMSE: semi-supervised federated learning approach for IoT network intrusion detection, Comput. Secur. 151 (2025) 104337. https://doi.org/10.1016/j.cose.2025.104337
[32] B. Sharma, L. Sharma, C. Lal, S. Roy, Explainable artificial intelligence for intrusion detection in IoT networks: a deep learning based approach, Expert Syst. Appl. 238 (2024) 121751. https://doi.org/10.1016/j.eswa.2023.121751
[33] W. Serrano, CyberAIBot: artificial intelligence in an intrusion detection system for CyberSecurity in the IoT, Future Gener. Comput. Syst. 166 (2025) 107543. https://doi.org/10.1016/j.future.2024.107543
[34] A. Nazir, J. He, N. Zhu, A. Wajahat, X. Ma, F. Ullah, S. Qureshi, M.S. Pathan, Advancing IoT security: a systematic review of machine learning approaches for the detection of IoT botnets, J. King Saud Univ. Comput. Inform. Sci. 35 (10) (2023) 101820. https://doi.org/10.1016/j.jksuci.2023.101820
[35] M. Keshk, N. Koroniotis, N. Pham, N. Moustafa, B. Turnbull, A.Y. Zomaya, An explainable deep learning-enabled intrusion detection framework in IoT networks, Inf. Sci. 639 (2023) 119000. https://doi.org/10.1016/j.ins.2023.119000
[36] F. Afifi, F. Zaki, H. Hanif, N. Aqil, N.B. Anuar, Transformer-based tokenization for IoT traffic classification across diverse network environments, PeerJ Comput. Sci. 11 (2025). https://doi.org/10.7717/peerj-cs.3126
[37] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP) (2018) 108–116. https://doi.org/10.5220/0006639801080116
[38] B. Bazaluk, M. Hamdan, M. Ghaleb, M.S.M. Gismalla, F.S.C. da Silva, D.M. Batista, Towards a transformer-based pre-trained model for IoT traffic classification, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), IEEE, 2024, pp. 1–7. https://doi.org/10.1109/NOMS59830.2024.10575448
[39] I. Vaccari, G. Chiola, M. Aiello, M. Mongelli, E. Cambiaso, MQTTset, a new dataset for machine learning techniques on MQTT, Sensors 20 (22) (2020). https://doi.org/10.3390/s20226578
[40] R. Kozik, M. Pawlicki, M. Choraś, A new method of hybrid time window embedding with transformer-based traffic data classification in IoT-networked environment, Pattern Anal. Appl. 24 (4) (2021) 1441–1449. https://doi.org/10.1007/s10044-021-00980-2
[41] M. Wang, N. Yang, N. Weng, Securing a smart home with a transformer-based IoT intrusion detection system, Electronics 12 (9) (2023). https://doi.org/10.3390/electronics12092100
[42] S.-M. Tseng, Y.-Q. Wang, Y.-C. Wang, Multi-class intrusion detection based on transformer for IoT networks using CIC-IoT-2023 dataset, Future Internet 16 (8) (2024). https://doi.org/10.3390/fi16080284
[43] C. Zhang, J. Li, N. Wang, D. Zhang, Research on intrusion detection method based on transformer and CNN-BiLSTM in internet of things, Sensors 25 (9) (2025). https://doi.org/10.3390/s25092725
[44] A.H. Lashkari, et al., CICFlowMeter (ISCXFlowMeter) V4.0: a network traffic flow generator and analyzer, 2018. Canadian Institute for Cybersecurity, UNB, https://github.com/ISCX/CICFlowMeter.