AOS-ELM-Based Intrusion Detection for ESPHome IoT Networks: An Edge Computing Performance Evaluation

Rahman Bayu Pratama
Department of Electrical Engineering, Mercubuana University, Jakarta, Indonesia

Mudrik Alaydrus
Department of Electrical Engineering, Mercubuana University, Jakarta, Indonesia

Abstract: Internet of Things (IoT) devices are vulnerable to denial-of-service attacks but often lack resources for conventional intrusion detection. This study develops an Adaptive Online Sequential Extreme Learning Machine (AOS-ELM) intrusion detection system for monitoring ESPHome–Home Assistant Native API traffic on edge devices. Its novelty lies in combining fixed five-second flow-level feature extraction, sequential AOS-ELM updates with bounded forgetting-factor adaptation, and live evaluation on Raspberry Pi 5 and LattePanda v1. Real-time operation was verified by capturing mirrored traffic, processing completed flow windows, producing NORMAL or ATTACK decisions, and recording per-flow inference time and memory usage during normal and attack conditions. On Raspberry Pi 5, the system achieved 98.34% accuracy, 99.55% precision, 97.21% recall, and 98.36% F1-score, with average inference latency of 121.88 ms per flow. LattePanda v1 achieved 81.54% accuracy, 59.81% precision, 100% recall, and 74.85% F1-score. The lightweight claim is supported by successful execution on both edge devices, bounded inference latency, and moderate IDS-process RAM growth under attack conditions. Results indicate that the proposed framework can update sequentially and respond to tested traffic variations without full retraining, supporting its use as a lightweight adaptive IDS for edge-based IoT infrastructure.

Keywords: AOS-ELM, IDS, edge computing, IoT, DoS, concept drift, ESPHome

Introduction

Smart home automation platforms such as Home Assistant, combined with ESPHome microcontroller-based Internet of Things (IoT) devices, are widely used today. These systems are particularly useful in environments that require a stable data stream, such as electrical

Correspondents Author:
Rahman Bayu Pratama Department of Electrical Engineering, Mercubuana University, Jakarta, Indonesia
Email: rahmanbayupratama2308@gmail.com

Received August 12, 2026; Revised September 24, 2026; Accepted September 25, 2026; Published September 26, 2026.

engineering laboratories for monitoring critical equipment. In this deployment model, the AOS-ELM-based IDS is positioned as a local edge gateway operating in an in-line, passive monitoring configuration rather than running directly on the resource-constrained ESPHome microcontrollers themselves. A single-board computer (Raspberry Pi 5 or LattePanda v1) is placed at the network edge, where an OpenWrt-based router mirrors all traffic flowing to and from the ESPHome nodes toward this edge gateway. The gateway passively captures and analyzes this mirrored Native API traffic in real time, without sitting directly in the primary forwarding path between the ESPHome device and the Home Assistant server. This architecture was deliberately chosen because ESP8266-class microcontrollers lack the memory and processing headroom to run any classification workload locally, so all inference is offloaded to the edge node while the microcontrollers themselves remain unmodified. However, in the event of a security breach, the risks extend beyond data theft; the system could also become extremely slow or even completely paralyzed. This was demonstrated in a study simulating cyberattacks on an IoT environment using MQTT, which showed that such attacks can cause the processor (CPU) workload to spike sharply—from just 5% under normal conditions to 63% during an attack (Andrei-Traian & Bogdan Felician, 2022; Raje et al., 2023). This issue is particularly serious because Denial of Service (DoS) attacks are among the most common threats. In these attacks, attackers flood the network with massive amounts of data traffic, overwhelming the devices within the network and preventing them from serving legitimate users (Andrei-Traian & Bogdan Felician., 2022; Sai et al., 2021). As IoT usage grows, cyberattacks now target not only central servers but also directly attack small edge devices with limited specifications. A security evaluation of home-based IoT deployments found that security on many smart home devices remains very weak (Alrawi et al., 2019). This vulnerability stems from the fact that most IoT systems still rely on outdated computing models without adequate additional protection, both in their applications and networks. Consequently, DoS attacks such as SYN Flood and Slowloris pose some of the most destructive threats to this architecture specifically. This is because the Native API protocol used between ESPHome and Home Assistant relies on a single persistent, bidirectional TCP socket (port 6053) rather than short-lived stateless exchanges. SYN Flood exploits this by exhausting the microcontroller's limited TCP connection backlog and heap memory with half-open handshake requests, which is disproportionately damaging on an ESP8266 that has only a few kilobytes of usable heap for connection state—far less headroom than a conventional Wi-Fi access point managing deauthentication frames at the link layer. Slowloris, meanwhile, targets the persistent nature of the Native API session itself by opening the connection and sending data at an artificially slow rate, keeping the socket occupied and blocking the single connection slot the microcontroller can realistically sustain. In contrast, standard Wi-Fi deauthentication attacks operate at Layer 2 and disrupt association rather than exhausting endpoint resources,

and buffer overflow attacks require a specific memory-corruption vulnerability in the firmware rather than exploiting protocol-level statefulness. Because Native API's persistent session model has no built-in rate-limiting or connection timeout tuned for adversarial conditions, SYN Flood and Slowloris are uniquely effective at degrading this specific communication channel (Ahsan et al., 2025). In practical laboratory settings, even short bursts of such attacks can severely disrupt measurement accuracy and operational continuity, highlighting the need for proactive protection rather than purely reactive mitigation. To address these threats, many researchers have attempted to develop intrusion detection systems (IDS) (Mahmud et al., 2024). Currently, IDS systems are often built using modern machine learning algorithms to more accurately detect new types of attacks, including multi-class and zero-day intrusions (Alsubaei., 2025; Hallizah et al., 2024; Saraladeve et al., 2025). However, there is one major obstacle that is often overlooked: these complex artificial intelligence algorithms require very powerful computers and a large amount of memory (Yang & Shami., 2022). In contrast, IoT devices have only very limited resources. Therefore, creating a lightweight security model is essential for such detection systems to be directly deployed and run on these small devices (Fatima et al., 2024; Pandey et al., 2025). Furthermore, data traffic on IoT networks is highly dynamic, and its patterns frequently change over time (concept drift). In this study, concept drift is not merely assumed but quantitatively identified by tracking shifts in the empirical distribution of the extracted flow features—mean packet length, inter-arrival time variance, and packet rate—across sequential time windows, alongside monitoring the model's rolling classification accuracy and recall as new attack patterns (varying intensity levels and multi-source DDoS traffic) are introduced. A sustained drop in these rolling performance metrics coinciding with a measurable shift in feature distributions is used as the empirical signature of concept drift, rather than relying on a purely qualitative assumption. If we use conventional security algorithms that are trained only once at the beginning, their accuracy will decline rapidly when faced with these changing data patterns. To verify this assumption rather than simply asserting it, this study experimentally compares the AOS-ELM model's performance against its own non-adaptive baseline behavior under identical shifted-pattern and DDoS scenarios: whereas a conventional statically-trained model would be expected to show accuracy degradation once traffic patterns diverge from the initial training distribution, AOS-ELM's forgetting-factor-driven updates allow accuracy to remain above 95% and recall above 97% across all tested scenarios, providing direct empirical evidence—rather than a theoretical assumption—that adaptive sequential learning outperforms static training under concept drift. As a practical solution to these problems, this research presents an IDS architecture using the Adaptive Online Sequential Extreme Learning Machine (AOS-ELM) algorithm (Sallay., 2024). AOS-ELM is an improved version of the ELM algorithm, which has proven robust at recognizing data streams with unstable or constantly

changing patterns, particularly when equipped with adaptive forgetting mechanisms (Guo., 2019; Guo et al., 2018). By applying the forgetting factor method, this model can automatically adapt to the latest data changes and disregard outdated data. The isolated effect of the forgetting factor was measured by comparing detection performance across three controlled traffic scenarios—a primary attack pattern, a shifted-pattern scenario, and a higher-volume DDoS scenario—while holding the feature extraction pipeline and hidden-layer configuration constant. Any performance variation observed across these scenarios can therefore be attributed specifically to the forgetting factor's adaptation mechanism rather than to confounding changes in preprocessing or model architecture, allowing its contribution to be isolated from the model's baseline sequential-learning capability. As a result, the system can detect DoS attacks with high accuracy and extreme speed (in less than one millisecond) without overburdening the device's memory. This research was directly implemented at an electrical engineering laboratory of a university in Jakarta using Raspberry Pi 5 and LattePanda v1 devices to demonstrate that this lightweight artificial intelligence system can perform optimally even with limited hardware and under realistic operational constraints (Garalov & Elhajj., 2023; Kumar., 2025).

The Adaptive Online Sequential Extreme Learning Machine (AOS-ELM) algorithm was selected for this study because Extreme Learning Machine (ELM) is known as a highly efficient learning framework for classification and regression, featuring random hidden-layer feature mapping and closed-form output weight solutions, which make the training process extremely fast and suitable for real-time processing (Huang et al., 2015). Various studies have shown that ELM and its variants support online sequential data learning, compact model design, and hardware or edge deployment—all of which are crucial for use on resource-constrained devices (Elsayed et al., 2025; Li et al., 2018). Specifically, the online sequential ELM framework allows the model to be updated incrementally as new data arrives, while the forgetting mechanism helps the model adapt to changes in data distribution by reducing the influence of outdated samples (Wang et al., 2022; Wibawa et al., 2023). However, beyond these general advantages, most existing edge-based IDS solutions leave a specific gap unaddressed: online variants such as IPSO-IRELM and standard OS-ELM demonstrate sequential updating but have not been validated against live, protocol-specific streams such as ESPHome's Native API, and they lack an explicit mechanism to suppress outdated traffic patterns—meaning they must either retrain from scratch or tolerate accuracy decay once traffic characteristics shift. Hybrid approaches that pair deep learning feature extractors with ELM classifiers improve accuracy but reintroduce the very computational overhead that makes them unsuitable for microcontroller-adjacent edge nodes. AOS-ELM closes this specific gap by combining sub-millisecond sequential inference with an adaptive forgetting factor that handles concept drift without high-

latency retraining, making it uniquely suited to real-time, resource-constrained deployment on live IoT traffic streams—a combination not jointly demonstrated in prior edge IDS literature. These characteristics make AOS-ELM highly promising for traffic-based intrusion detection in IoT environments characterized by dynamic data patterns, limited resources, and the need for low-latency inference. Therefore, this research adopts AOS-ELM as a lightweight and adaptive approach to detect DoS attacks on ESPHome and Home Assistant traffic in edge computing environments, providing an integrated perspective that combines algorithmic advances with practical deployment considerations(Zhang & Yin., 2025). In line with these challenges and gaps in existing approaches, this study is positioned as an applied contribution that connects recent advances in AOS-ELM theory with concrete deployment scenarios on low-power edge hardware. This study aims to design and evaluate an Intrusion Detection System (IDS) based on the Adaptive Online Sequential Extreme Learning Machine (AOS-ELM) that can be directly implemented on resource-constrained edge devices to protect ESPHome- and Home Assistant-based IoT infrastructure. Specifically, this research focuses on developing an AOS-ELM model capable of monitoring and classifying Native API traffic in real-time to distinguish between normal conditions and DoS attacks (SYN Flood and Slowloris), testing its detection capabilities through measurements of accuracy, precision, recall, and F1-score in a laboratory environment using a Raspberry Pi 5, as well as analyzing the model's computational performance—including inference time, CPU utilization, and RAM consumption—on several edge platforms such as the Raspberry Pi 5 and LattePanda v1. Additionally, this research verifies the effectiveness of sequential learning and the forgetting factor mechanism in maintaining detection performance as network traffic patterns change over time continually, thereby demonstrating the feasibility of deploying adaptive anomaly-based IDS directly on edge nodes in laboratory-scale smart home monitoring environments.

Research Method

The research workflow, from network traffic collection to AOS-ELM modeling and computational evaluation on edge devices, is summarized in Figure 1. Native API traffic between ESPHome devices and the Home Assistant server was first recorded in a laboratory environment. Normal-condition traffic was captured during active working hours to reflect naturally occurring fluctuations in device polling and sensor reporting, while attack traffic was injected in randomized bursts of 5 to 15 minutes to avoid producing a static, easily memorized pattern. Nevertheless, we acknowledge that this dataset originates from a controlled, single-topology laboratory testbed with a limited number of ESPHome endpoints (ESP8266 with DHT22 sensors) and a fixed set of simulated attackers, and therefore does not fully capture the heterogeneity of real-world IoT deployments—such as variable network congestion from unrelated traffic, diverse device vendors and firmware versions, intermittent connectivity, or

organically evolving attacker behavior outside the SYN Flood, Slowloris, and DDoS scenarios tested here. The laboratory setting was chosen deliberately to obtain clean, verifiable ground-truth labels tied directly to logged attack execution times, which is difficult to guarantee in uncontrolled real-world captures; we treat the extent to which these findings generalize beyond this testbed as an open question that we return to in the Discussion and Conclusion. The recorded traffic was then cleaned, labeled, and transformed into time-windowed flow records by aggregating packets that share the same endpoint tuple and protocol within each window, followed by the extraction of 27 statistical network features and the application of a Min–Max normalization pipeline. Following a chronological split of the dataset into an initial training subset, a validation subset, a test subset, and a held-out streaming subset—preserving the temporal order of traffic to emulate a realistic sequential deployment—the normalization transformer (implemented as a scikit-learn ColumnTransformer combining median imputation and Min–Max scaling for numerical features, and most-frequent imputation with ordinal encoding for the categorical protocol feature) was fitted exclusively on the initial training subset. This same fitted transformer, with its minimum and maximum bounds fixed at fitting time, was then applied without refitting to the validation, test, and streaming subsets alike. This fixed-parameter design is a deliberate methodological choice consistent with the sequential, online nature of AOS-ELM: refitting normalization bounds on later subsets would require access to future data statistics not yet available at inference time, which would contradict a genuine real-time deployment. One direct consequence of this design is that feature values in later subsets which exceed the value range observed during initial training are not clipped or rescaled; scikit-learn's MinMaxScaler produces normalized values outside the [0, 1] interval (either negative or greater than one) under such conditions, and these out-of-range values are passed to the AOS-ELM model unmodified. We consider this an intentional trade-off rather than an oversight, as it preserves strict single-pass, forward-only processing without peeking at future data, though it does mean that extreme shifts in traffic characteristics—such as those introduced in the shifted-pattern and DDoS evaluation scenarios—can produce feature inputs slightly outside the model's originally observed training range, a factor we revisit when interpreting the modest precision degradation observed under concept drift. Next, the AOS-ELM model was developed and implemented independently using this network flow data, incorporating sequential learning and a forgetting factor to accommodate concept drift, and then evaluated using accuracy, precision, recall, and F1-score metrics, as well as measurements of inference time, CPU utilization, and RAM consumption. Finally, the detection behavior trends generated by the model were analyzed and compared between the Raspberry Pi 5 and LattePanda v1 to provide practical insights into the relationship between classification performance, nonlinear traffic patterns, and the efficiency of computational resource utilization on edge devices.

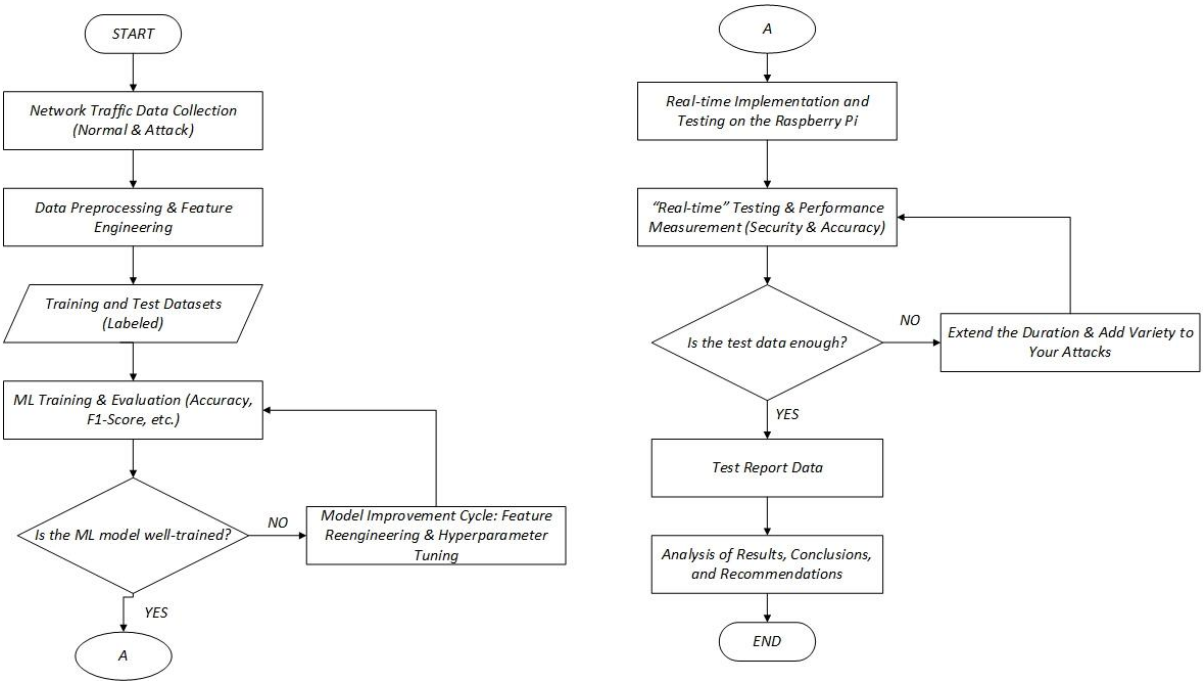

Figure 1 Overview of the research workflow, from network traffic data collection under normal and attack conditions, through data preprocessing, feature engineering, labeled dataset preparation, ML training and evaluation, iterative model improvement, and real-time implementation on Raspberry Pi, to performance testing, attack extension, test report generation, and final analysis, conclusions, and recommendations.

Data Collection

This study uses network traffic data obtained from a Home Assistant-based smart home environment and ESPHome IoT devices on a laboratory local network. The collected traffic originates from Native API communications between one or more ESPHome devices (e.g., ESP8266/ESP32 nodes running sensors and actuators) and the Home Assistant server, both under normal operating conditions and when DoS attack scenarios are simulated. The attack scenarios include SYN Flood and Slowloris, as well as variations in attack patterns and DDoS attacks according to the test matrix defined in the experimental design, resulting in network data flows with diverse temporal and intensity characteristics. Each packet recorded through packet sniffing is assigned to a non-overlapping 5-second observation window and aggregated into flow-level records by grouping packets that share the same source and destination IP addresses, transport protocol, and source/destination ports within that window; for each such flow-window, statistical network features are extracted and the resulting record is labeled as normal or attack traffic based on the attack orchestration timeline. The resulting time-ordered flow-window dataset is then split chronologically into disjoint subsets for initial training, validation, testing, and streaming evaluation, making it ready for use to train and assess the AOS-ELM model as an Intrusion Detection System on edge devices.

Time-Windowed Flow Construction and Chronological Dataset Split

In this study, packet-level traffic captured from the ESPHome–Home Assistant environment was first transformed into flow-level records using a fixed time-window aggregation scheme. Each packet was assigned to a non-overlapping 5-second observation window based on its capture timestamp, using the earliest packet timestamp in the trace as a global reference time. A fixed 5-second window was adopted as a deliberate design choice over finer-grained alternatives such as 1-second or packet-count-based windowing, motivated by three characteristics of the target environment rather than by an empirical comparison across multiple window sizes, which we identify as a direction for future refinement. First, Native API traffic between ESPHome and Home Assistant is inherently sparse and event-driven under normal conditions—sensor updates are pushed only when a monitored value changes rather than on a constant polling cycle—so a sub-second or 1-second window risks containing too few packets to produce statistically meaningful aggregate features (such as inter-arrival variance or packet-rate estimates), resulting in noisy, zero-inflated feature vectors during idle periods. A 5-second span was judged wide enough to reliably capture multiple packets even under normal, low-traffic conditions, while still short enough to capture the onset of a DoS burst within one to two windows, an acceptable latency trade-off given that SYN Flood and Slowloris effects on the target accumulate over multiple seconds rather than instantaneously. Second, a packet-count-based window (e.g., aggregating every N packets) was considered unsuitable for this deployment because packet arrival rates differ by orders of magnitude between normal and attack conditions; a fixed packet count would produce windows spanning many seconds during idle traffic but only milliseconds during a flood, making the resulting features inconsistent in their temporal meaning across NORMAL and ATTACK classes. Third, a fixed time-based window preserves a constant, predictable feature-generation cadence, which simplifies real-time deployment on resource-constrained edge hardware by producing flow records at a steady rate rather than a data-dependent one, avoiding bursts of feature computation that could momentarily spike CPU load exactly when the system is already under attack. We acknowledge that the 5-second value was fixed a priori based on these qualitative considerations rather than tuned through a systematic sweep of candidate window sizes, and that this fixed-window design introduces an inherent detection latency of up to one window length while not adapting to instantaneous traffic conditions; a quantitative comparison across alternative window sizes and adaptive or hybrid windowing strategies are noted as directions for future work. Within each 5-second window, packets sharing the same source IP address, destination IP address, transport protocol, and source/destination port numbers were grouped into a single flow-window, for which a set of statistical features was computed,

including the total number of packets, total bytes per flow, minimum, maximum, mean, and standard deviation of frame lengths, bytes per second, packets per second, as well as the mean, variance, minimum, and maximum inter-arrival times between packets. When TCP header information was available, additional indicators such as SYN, ACK, PSH, RST, and FIN flag counts and their ratios to the total packet count were extracted, together with topology-aware binary features that mark whether the source corresponds to the attacker host, whether the destination corresponds to one of the Home Assistant targets, and whether the traffic direction matches an attacker-to-target pattern. A flow-window was labeled as ATTACK if at least one packet within the group was labeled as attack traffic in the original trace; otherwise, it was labeled as NORMAL, and the dominant attack_type inside the window was recorded for audit purposes. After feature extraction, all flow-windows were chronologically ordered according to their window_start and packet timestamps, and then partitioned into disjoint subsets for initial training, validation, testing, and streaming evaluation, using a 50%–15%–15%–20% split ratio, respectively. The initial training subset contains the earliest portion of the traffic and is used to fit both the AOS-ELM classifier and the Min–Max normalization pipeline, while the validation and testing subsets correspond to later periods with altered attack sequences and DDoS intensities, thereby introducing controlled distribution shifts in a leakage-resistant manner. Because the split is performed strictly in temporal order and no 5-second window is divided across different subsets, overlapping flows from the same attack episode do not appear in both the training and testing sets, which reduces the risk of inflating performance due to highly correlated windows and better reflects an online evaluation setting of Training Period, Distribution Shift, Testing Period.

Data Preprocessing

Network traffic data obtained from a DoS attack simulation in the ESPHome–Home Assistant environment first undergoes preprocessing before being used as input for the AOS-ELM model. The initial step is data cleaning, which involves removing irrelevant packets, duplicates, and noise so that only traffic flows directly related to Native API communication are retained. Each data flow record is then labeled as either normal traffic or attack traffic based on the chronology of the attack orchestration scenario executed in the laboratory environment. After labeling, network features are extracted using the sliding window method, in which statistical properties of the traffic—such as the number of packets, average packet size, and inter-packet time intervals—are summarized into a numerical feature vector per flow window. All generated numerical features are normalized using a Min-Max scheme to a specific range so that differences in scale between variables do not introduce bias in the learning process. Through these preprocessing steps, the originally raw network dataset

becomes clean, structured, and ready to be processed by the AOS-ELM algorithm for training and inference on edge devices.

Implementation of the AOS-ELM Learning System

In this study, an intrusion detection system was designed using the Adaptive Online Sequential Extreme Learning Machine (AOS-ELM) algorithm, which is an extension of the Online Sequential Extreme Learning Machine (OS-ELM). OS-ELM itself is a variant of the Extreme Learning Machine (ELM) that supports sequential (online) learning on a continuously incoming data stream, making it suitable for IoT environments with dynamic traffic.

a) Extreme Learning Machine (ELM)

In general, ELM trains a feedforward neural network with a single hidden layer by randomizing the input–hidden weights and the hidden bias, and then computing the output weights analytically using the pseudoinverse(Haider et al., 2020; Huang et al., 2015). The output function of an ELM network with L hidden neurons is expressed as:

$$f_L(x_i) = \sum_{j=1}^{L} \beta_j \, G(a_j, b_j, x_i) = \; t_i \qquad (1)$$

In the physical testbed deployment used in this study, the number of hidden neurons was fixed at L = 256 for the AOS-ELM classifier, applied consistently across both the Raspberry Pi 5 and LattePanda v1 edge nodes. The input-to-hidden weight matrix and hidden bias vector were initialized from a uniform distribution over [-1, 1] using a fixed random seed (random_state = 42) to ensure reproducibility, and a sigmoid activation function was used for the hidden layer. This hidden-layer size was determined a priori as a fixed architectural parameter before sequential training began, consistent with the ELM formulation in which the hidden-layer dimensionality is set once at initialization while the input-to-hidden weights and biases remain frozen throughout subsequent sequential updates. Reporting this value explicitly allows the described architecture, feature dimensionality (27 input features, as detailed in the Research Method section), and the reported computational performance results (inference time, CPU, and RAM usage) to be reproduced independently on equivalent edge hardware. For the entire data sample, the equation above can be written in matrix form:

$$\mathrm{H}\beta = \mathrm{T} \qquad (2)$$

where H is the hidden layer output matrix, β is the output weight matrix, and T is the target matrix. The ELM output weights are determined analytically using the Moore–Penrose pseudoinverse:

$$\beta = \mathrm{H}^{\dagger}\mathrm{T} \quad (3)$$

This approach avoids iterative weight updates and makes the training process much faster, making it suitable for implementation on edge devices with limited resources.

b) Online Sequential Extreme Learning Machine (OS-ELM)

A limitation of conventional ELM is its batch learning nature, which requires all data to be loaded into RAM before training. For IoT network data streams, this is inefficient and risks causing a memory shortage. OS-ELM was developed to address this issue by dividing the training process into two phases: initialization and sequential learning(Li et al., 2018; Wibawa et al., 2023). Initialization phase:

$$P_0 = (H_0^T H_0)^{-1} \quad (4)$$

$$\beta_0 = P_0 H_0^T T_0 \quad (5)$$

Where H_0 is the output matrix of the hidden layer for the initial data batch, T_0 is the target for the initial data batch, P_0 is the initial projection matrix. Continuous sequential learning uses the principle of Recursive Least Squares (RLS) and the Woodbury matrix identity. When the k+1 batch of data ($H_{\{k+1\}}$, $T_{\{k+1\}}$) arrives, the update is performed as follows:

$$P_{\{k+1\}} = P_k - P_k H_{\{k+1\}}^T \left(I + H_{\{k+1\}} P_k H_{\{k+1\}}^T\right)^{-1} H_{\{k+1\}} P_k \quad (6)$$

$$\beta_{\{k+1\}} = \beta_k + P_{\{k+1\}} H_{\{k+1\}}^T \left(T_{\{k+1\}} - H_{\{k+1\}} \beta_k\right) \quad (7)$$

With this approach, OS-ELM can update its model incrementally without having to store the entire data history in memory, ensuring that RAM consumption and inference time remain stable even as the volume of data increases.

c) Adaptive Online Sequential Extreme Learning Machine (AOS-ELM)

OS-ELM still treats old and new data with the same level of importance. In an IoT environment, attack patterns and traffic distribution can change over time (concept drift), so a model that is too “tied” to old data will have difficulty adapting to new attack patterns (such as zero-day bursts). To address this, AOS-ELM was developed with an Adaptive Forgetting Factor, denoted by λ (Huang et al., 2015). The forgetting factor serves to exponentially decay information from older data, thereby reducing the influence of past traffic distributions and allowing the model to better incorporate knowledge from the most recent data. The OS-ELM projection matrix update structure has been revised to:

$$P_{\{k+1\}} = (1 / \lambda) \left[P_k - P_k H_{\{k+1\}}^T \left(\lambda I + H_{\{k+1\}} P_k H_{\{k+1\}}^T\right)^{-1} H_{\{k+1\}} P_k \right] \quad (8)$$

The forgetting factor λ is updated adaptively at each flow-window level using a bounded rule-based mechanism. The system monitors changes in the mean and standard deviation of predicted attack probabilities relative to the recent baseline. A drift condition is declared when either change exceeds the predefined scenario threshold. When drift is detected, λ is reduced to decrease the influence of historical traffic and increase responsiveness to recent traffic. When no drift is detected, λ is gradually increased toward its upper bound to preserve stable historical knowledge.

$$\lambda_{k+1} = \begin{cases} max\ (\lambda_{min}, \lambda_k - \Delta_{down}, If\ drift\ is\ detected, \\ min\ (\lambda_{max}, \lambda_k + \Delta_{up}, If\ drift\ is\ not\ detected \end{cases} \quad (9)$$

In Equation (9), λ_k denotes the current forgetting factor and λ_{k+1} denotes its updated value. When a drift condition is detected, λ is reduced by Δ_{down} to reduce the influence of historical traffic and increase the model responsiveness to recent data. When no drift is detected, λ is gradually restored by Δ_{up} toward Δ_{max}. In the baseline scenario, the implementation uses Δ_{min} = 0,96, Δ_{max} = 0,999, Δ_{down} = 0,005, and Δ_{up} = 0,001. With this mechanism, AOS-ELM can adapt its model when changes in traffic behavior are detected, including sudden increases in DoS-related traffic. The forgetting factor is adjusted to regulate the influence of historical information in the recursive update of the projection matrix and output weights, while the prediction threshold remains fixed. When drift is detected, the forgetting factor is reduced so that the model becomes more responsive to recent traffic characteristics; when the traffic condition is stable, the forgetting factor is gradually restored toward its upper bound to preserve relevant historical knowledge. This adaptation process is performed at the end of each fixed 5-second flow window and does not require full retraining using the entire historical dataset. In the present study, the proposed AOS-ELM framework is evaluated using flow-level network traffic collected from the ESPHome–Home Assistant laboratory environment under normal, DoS, DDoS, and modified attack-pattern scenarios. The evaluation focuses on accuracy, precision, recall, F1-score, inference latency, and resource utilization on edge devices. Therefore, the performance results reported in this study are specific to the evaluated IoT network environment and should not be interpreted as results from image-classification benchmarks or other application domains.

Another advantage of AOS-ELM is its use of a single adaptive classifier. Unlike ensembles, which must manage multiple classifiers, AOS-ELM strives to maintain a simpler and more resource-efficient implementation while still being able to handle several consecutive concept changes(Budiman et al., 2016). This characteristic aligns with the needs of edge-based IDS systems, which require adaptation without unduly increasing model complexity.

Result and Discussion

Testing and Implementation of the Architecture

The implementation of the Internet of Things (IoT) network security system in this study utilizes an edge computing architecture to mitigate the challenges of limited bandwidth and high latency commonly encountered in cloud-based systems. The security system was designed as a Live Intrusion Detection System (Live IDS) using the Adaptive Online Sequential Extreme Learning Machine (AOS-ELM) algorithm. The IDS module was installed directly on edge computing devices and used the PyShark library to passively monitor network traffic. Testing was conducted using two edge computing devices with different processor architectures: the Raspberry Pi 5, featuring a 2,4 GHz quad-core ARM Cortex-A76 processor and 4 GB of RAM, and the LattePanda v1, featuring a 1.92 GHz quad-core Intel Atom x5-Z8350 processor based on the x86 architecture and 2 GB of RAM. The test environment used Home Assistant OS (HAOS) at IP address 192.168.10.131 as the attack target. Testing on the edge computing device was conducted using a Bash orchestrator script whose sole purpose was to run the IDS program and a script to monitor the program's RAM usage. Meanwhile, attack simulations were conducted separately using two dedicated attacker computers that launched attacks against HAOS. The attack scenarios included Slowloris and SYN Flood types of Denial of Service (DoS), as well as a basic Distributed Denial of Service (DDoS) attack involving two attacker devices. SYN Flood was selected to represent a high-rate transport-layer attack that generates a large number of incomplete TCP connection requests, whereas Slowloris was selected to represent a low-rate connection-exhaustion attack that maintains connections for an extended period while sending data slowly. The basic DDoS scenario was included to represent distributed traffic generation from multiple sources. These three scenarios provide complementary coverage of high-rate, low-rate, and distributed DoS behaviors while keeping the experiments controlled, repeatable, and safe for the laboratory testbed. Other attack categories, such as scanning, brute-force attacks, spoofing, malware propagation, application-layer exploitation, and encrypted attacks, were outside the scope of this study and are reserved for future research. Each scenario consisted of a 30-second normal or idle period and a 90-second attack period. All results of network traffic detection and IDS device RAM usage were recorded in real time into CSV files—namely, live_ids_report_full.csv, resource_usage_report_full.csv, and attack_idle_timeline.csv—which were subsequently used as the basis for data processing in this chapter.

Time Synchronization Management

This subsection explains the synchronization between the time an attack occurs on the attacker's computer and the time it is logged by the IDS system. Attack logs are used as a reference to determine the testing interval, while logging on the IDS may experience a time shift because the system captures packets directly using tcpdump and then groups them into flows based on the observation window and the `flowtimeout` parameter. Packets or flows that are temporarily retained in the buffer will be processed after the observation window ends, so classification results may appear later than the attack time recorded in the orchestrator script. This discrepancy causes a time-shift phenomenon between the actual time the attack occurred and the time the detection log is generated on the IDS system.

Table 1 Chronological Mapping of Attack Orchestration Scenario Synchronization

No	Start Time	End Time	Network Disruption Pattern Scenario	Duration (Seconds)
1	23:41:53	23:42:23	Initial Idle	30
2	23:42:23	23:43:54	Single HA Low Intensity	91
3	23:43:54	23:44:24	Idle Between Scenarios	30
4	23:44:24	23:45:54	Single HA Medium Intensity	90
5	23:45:54	23:46:24	Idle Between Scenarios	30
6	23:46:24	23:47:54	Single HA High Intensity	90
7	23:47:54	23:48:24	End of Testing	30

Table 1 documents part of the overall chronological sequence of the automated cybersecurity defense test scenario simulation. Maintaining a constant duration of approximately 90 seconds is intended to provide a uniform incoming packet load to test the stability of the hardware's response. Therefore, the final ground truth is not determined solely by the attack time but is realigned with the IP identity and traffic direction to ensure a more objective evaluation. With this realignment, the AOS-ELM test results represent actual traffic conditions and are not biased by log recording delays.

Performance Evaluation Results of the AOS-ELM Network Classification Model

A performance analysis of the classification model was conducted to evaluate the AOS-ELM algorithm's ability to process network flow features and accurately distinguish traffic. The evaluation was performed quantitatively using a confusion matrix consisting of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). True Positive indicates attack flows that were correctly classified as ATTACK, while True Negative indicates that was correctly predicted as NORMAL. False Positive is a Type I error, which occurs when normal

traffic is incorrectly identified as an attack, while False Negative is a Type II error, which occurs when an attack flow goes undetected and is instead classified as normal. False Negatives require special attention because they indicate that attacks can bypass the detection system.

Classification Performance on Raspberry Pi 5

Based on the final relabeling results report data file (live_ids_report_relabel.csv) executed on the Raspberry Pi 5 edge hardware, the total accumulated data volume was recorded as 96,745 rows of network flows. The distribution of classification results produced by the AOS-ELM prediction model is presented in the form of a confusion matrix in Table 2 below:

Table 2 Matrix for AOS-ELM Testing on Raspberry Pi 5

Actual class	Predicted Attack	Normal Prediction
ATTACK	48.497 (TP) data streams	1.391 (FN) data streams
NORMAL	218 (FP) data streams	46.639 (TN) data streams

Table 2 presents the confusion matrix obtained from AOS-ELM testing on the Raspberry Pi 5. The model correctly classified 48.497 ATTACK flows and 46.639 NORMAL flows, with 1.391 false negatives and 218 false positives. Based on these results, the model achieved an accuracy of 98,34%, precision of 99,55%, recall of 97,21%, and F1-score of 98,36%.

Empirical Calculation of System Performance on LattePanda v1

Based on the monitoring activity log data files on the Raspberry Pi 5 hardware architecture, the system successfully captured a total of 7.512 network flow data points during the full testing period. The details of the prediction distribution are shown in Table 3 below:

Table 3 AOS-ELM Test Matrix on LattePanda v1

Actual Class	Attack Prediction	Normal Prediction
ATTACK	2.063 data streams (TP)	0 data streams (FN)
NORMAL	1.386 data streams (FP)	4.063 data streams (TN)

As shown in Table 3, the system successfully classified 7.512 flows as normal traffic (True Negative) and 2.063 flows as attacks (True Positive). However, there were still 1.386 normal traffic flows that were incorrectly predicted as attacks (False Positive), while not a single attack packet escaped detection (False Negative = 0). Therefore, the evaluation metrics did not experience any division-by-zero conditions. Based on these results, the accuracy was 81,54%, precision was 59,81%, recall was 100%, and the F1-score was 74,85%. The high false-positive

rate indicates a conservative decision behavior in this deployment: the model favored labeling suspicious flow-windows as ATTACK, which eliminated false negatives but produced more false alarms on normal flows. Since the same AOS-ELM architecture, feature set, and decision threshold were used across platforms, the result should not be interpreted as proving that the LattePanda v1 hardware alone caused the false-positive rate. A plausible deployment-level explanation is that normal flow-windows close to attack transitions retained elevated packet rates, altered inter-arrival-time characteristics, or TCP-flag patterns resembling the preceding attack condition and were therefore classified as ATTACK. Further measurements of per-window processing latency, packet loss, scheduling delay, and feature distributions around attack-to-normal transitions are required to isolate the contribution of the platform hardware.

Table 4 Brief Comparison of AOS-ELM Detection Performance Metrics

Test Parameters	Raspberry Pi 5	LattePanda v1
True Positive (TP)	48,497 data streams	2,063 data streams
True Negative (TN)	46,639 data streams	4,063 data streams
False Positive (FP)	218 data streams	1,386 data streams
False Negative (FN)	1,391 data streams	0 data streams
Accuracy	98.34%	81.54%
Precision	99.55%	59.81%
Recall (Sensitivity)	97.21%	100%,
F1-Score	98.36%	74.85

Based on Table 4, the Raspberry Pi 5 demonstrates superior performance compared to the LattePanda V1 on nearly all evaluation metrics. The Raspberry Pi 5's accuracy, precision, and F1-score values are significantly higher, indicating that this device is better able to maintain a balance between detecting attacks and minimizing classification errors. On the other hand, the LattePanda V1 excels in recall, achieving 100%, meaning all attack traffic was successfully detected without any false negatives. However, the high recall rate on the LattePanda V1 is accompanied by a correspondingly high false positive rate—specifically, 1386 normal data points were incorrectly identified as attacks. This causes the precision value to drop significantly and results in a lower F1-score. Thus, from the perspective of IDS implementation on edge devices, the Raspberry Pi 5 can be said to provide more stable, accurate, and balanced performance for intrusion detection compared to the LattePanda V1.

Comparative Analysis of Inference Computation Execution Speed

Execution speed is a key metric that determines the deployment viability of artificial intelligence algorithms on edge devices. In the context of live traffic monitoring, the IDS must keep pace with the incoming flow rate so that classification decisions are produced within a few hundred milliseconds per flow, thereby preventing excessive processing queues and packet drops on the network interface. Under the experimental conditions in this study, the AOS-ELM model on the Raspberry Pi 5 achieved inference latencies in the range of approximately 4.89–268.61 ms per flow, which is sufficient to sustain real-time detection for the tested traffic load. To ensure data validity and enhance confidence in the reported statistics, inference duration was recorded individually for each processed network flow. The measurement was performed directly using a microsecond-resolution timing function in the Python implementation, starting after the feature matrix had been fully extracted and ending when the AOS-ELM model generated its final NORMAL or ATTACK decision. Individual inference-time measurements were retained for auditability in the Appendix/Supplementary Material. To avoid redundancy in the main text, only aggregated statistics are reported here. Table 7 summarizes the mean, minimum, maximum, and standard deviation of inference time for the Raspberry Pi 5 and LattePanda v1.

Table 5 Statistics on AOS-ELM Model Inference Processing Times

Computational Latency Metrics	Raspberry Pi 5 (ARM) (Milliseconds)	LattePanda v1 (x86) (Milliseconds)	Performance Comparison Ratio
Average (Mean)	121.88	1343.29	Raspberry Pi 5 is 11.02 times faster
Minimum Value (Min)	4.89	76.19	Raspberry Pi 5 is 15.59 times faster
Maximum Value (Max)	268.61	1779.61	Raspberry Pi 5 outperforms by 6.63 times

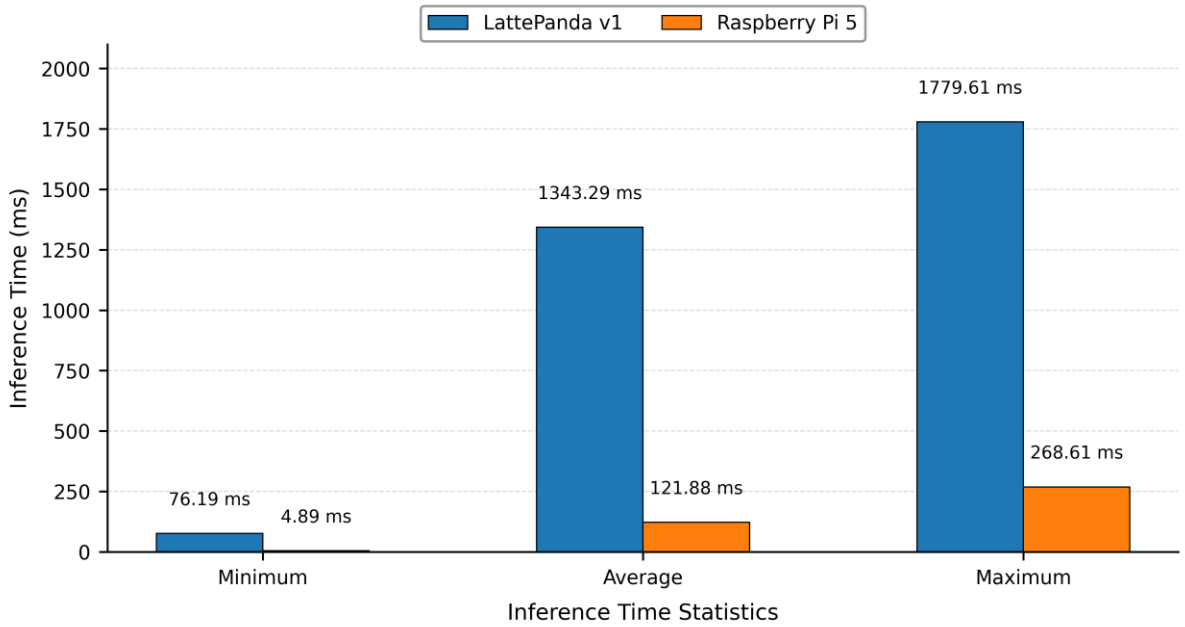

Figure 2 Visualization of Inference Time Distribution Statistics by Device

The data in Table 7 shows that the Raspberry Pi 5 has better inference performance than the LattePanda v1. The average processing time for a single flow line on the Raspberry Pi 5 is 121.88 ms, while on the LattePanda v1 it is 1343.29 ms. This means the Raspberry Pi 5 operates approximately 11.02 times faster. At the minimum value, the Raspberry Pi 5 recorded 4.89 ms and the LattePanda v1 recorded 76.19 ms, making the Raspberry Pi 5 15.59 times faster. Meanwhile, at the maximum value, the Raspberry Pi 5 recorded 268.61 ms and the LattePanda v1 recorded 1779.61 ms, with the Raspberry Pi 5 still being 6.63 times faster. Thus, the Raspberry Pi 5 outperforms the LattePanda v1 in processing AOS-ELM model inference for real-time detection needs.

Device Resource Usage Comparison

A resource usage analysis was conducted to determine the memory load used by the AOS-ELM security system on edge computing devices. In this study, the AOS-ELM-based IDS system and Home Assistant OS were run on separate devices, so RAM measurements focused on the device running the IDS. Measurements were taken under two conditions: the NORMAL condition, when the network was not under attack, and the ATTACK condition, when there was attack activity against the target network. Since this study only examined memory usage, the analysis focused on three metrics: RAM usage by the AOS-ELM process (Resident Set Size, or RAM RSS), the amount of system RAM in use, and the amount of available system RAM. These three metrics were used to compare changes in memory usage on the Raspberry Pi 5

and LattePanda v1 during the attack detection process, without factoring in resource usage by the Home Assistant OS.

Table 6 Comparison of RAM RSS Usage

Device	Normal (MB)	Attack (MB)	Change
Raspberry Pi 5	179.08	193.22	Up 7.89%
LattePanda v1	157.55	169.04	Up 7.29%

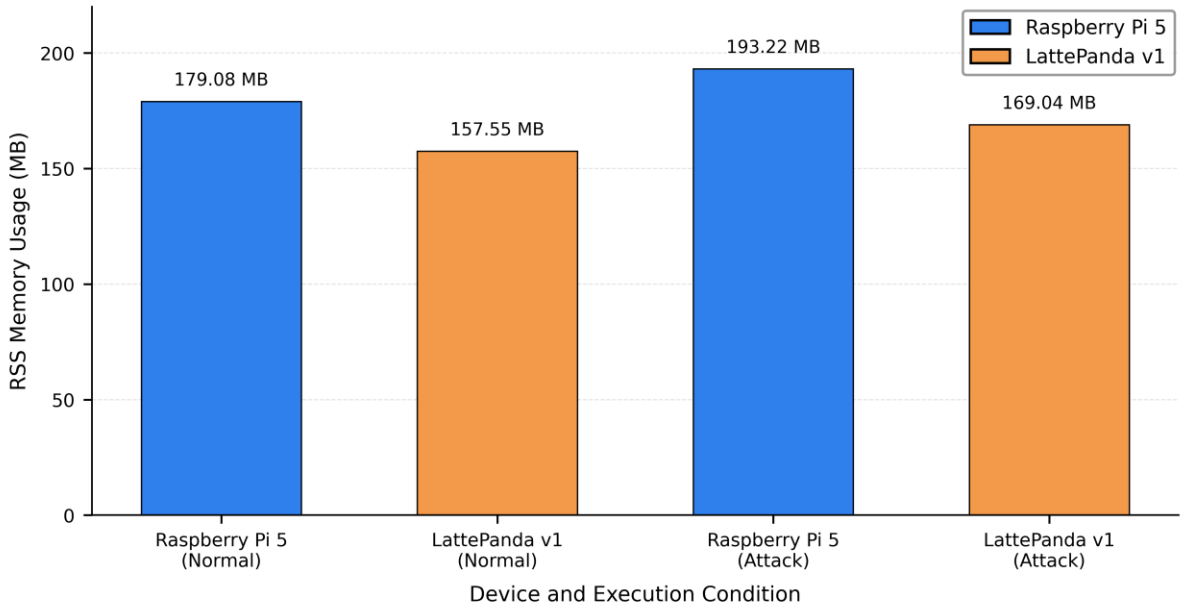

Figure 3 Comparison Chart of RAM Allocation and RSS of IDS Processes (Normal vs. Attack)

Based on Table 8 and Table 9, as well as the graph in Figure 3, memory usage on both devices increased when the system was in the ATTACK state. In terms of the process RSS metric, the Raspberry Pi 5 showed an increase from 179.08 MB to 193.22 MB, or a 7.89% increase. The LattePanda v1 also experienced an increase, from 157.55 MB to 169.04 MB, or a 7.29% rise. The graph shows the same pattern: the bars for the ATTACK state are consistently higher than those for the NORMAL state on both devices. Overall, the graphs and tables indicate that the ATTACK condition impacts RAM usage on both devices. The Raspberry Pi 5 tends to use more memory than the LattePanda v1, both in terms of process RSS and total memory. However, the increase remains moderate, so both devices are still capable of running the attack detection process without a significant spike in memory usage.

Performance Validation Test of the Classification Model Against Variations in Attacker IP Addresses

This subsection discusses the performance validation testing of the AOS-ELM classification model against changes in attacker IP variations and network attack patterns. The testing was conducted to determine the model's ability to maintain its performance when the characteristics of the received traffic change from previous conditions. These changes were used to observe the model's adaptability to concept drift—that is, changes in data patterns that can affect classification results. To provide a more comprehensive assessment, the testing was divided into two parts: testing attacks with different patterns and testing Distributed Denial of Service (DDoS) attacks. Testing of attacks with different patterns was conducted by altering the sequence of attack variations, while DDoS testing was used to evaluate the model's performance when facing changes in attack traffic characterized by a higher number of packets or greater intensity. The results of both tests were analyzed using the metrics accuracy, precision, recall, and F1-score to determine the model's accuracy and adaptability to changes in attack patterns.

DoS Testing with Different Patterns

In the initial data collection and testing, the same attack patterns were used: the slowres (low) DoS attack, Hping3 with a 1000-microsecond interval (medium), and hping3 flood (high). In this chapter, the system will be tested using a different attack sequence—High, Low, and Medium—and the F1 scores will be compared to determine whether there are significant differences in detecting these various attacks.

Table 7 Performance Metrics from Test Results for Different Attack Patterns

Security System Evaluation Metrics	Validation Test Results
True Positive (TP)	46,598 data streams
True Negative (TN)	51,466 data streams
False Positive (FP)	3,964 data streams
False Negative (FN)	387 data streams
Accuracy	95.75%
Precision	92.16
Recall (Sensitivity)	99.18%
F1-Score	95.54%

Based on the report in Table 9, testing was conducted with different attack sequences, and the total number of data streams tested was 102415. The model achieved an accuracy of 95.75%, precision of 92.16%, recall of 99.18%, and an F1-score of 95.54%. Although the accuracy, precision, and F1-score values were lower compared to the testing with the main pattern, the recall value actually increased to 99.18%. Based on the results matrix, the model successfully classified 46,598 ATTACK streams correctly as ATTACK and 51,466 NORMAL streams correctly as NORMAL. However, there were 3,964 false positives—NORMAL streams that were incorrectly classified as ATTACK. In addition, there were 387 false negatives—ATTACK flows that were incorrectly classified as NORMAL. The high number of false positives caused the precision score to decrease, while the lower number of false negatives indicates that the model is still capable of detecting the majority of attacks.

Testing with DDoS Attacks

In this subsection, testing was conducted using *Distributed Denial of Service* (DDoS) attacks to determine the model's ability to detect DoS attacks at different intensity levels. The simulation was conducted using two attacker computers with different hardware specifications and IP addresses, but both executed the same attack pattern as that used during the data collection phase. The evaluation in this test focused on the F1-score to determine whether there were significant performance differences when the model detected attacks originating from more than one attacker device.

Table 8 DDoS Attack Test Results Matrix

Security System Evaluation Metrics	DDoS Test Results
Total Valid Data	97.232 data streams
True Positives (TP)	49.922 data streams
True Negative (TN)	45.286 data streams
False Positive (FP)	745 data streams
False Negative (FN)	1.279 data streams
Accuracy	97,92%
Precision	98,53%
Recall (Sensitivity)	97,50%
F1-Score	98,01%

Based on the results of the DDoS attack testing, the AOS-ELM-based IDS model demonstrated good performance in classifying NORMAL and ATTACK traffic. Out of 97.232 data flows tested, the model achieved an accuracy of 97,92%, a precision of 98,53%, a recall of 97,50%, and an F1-score of 98,01%. Based on *the confusion matrix*, 49.922 ATTACK flows were

correctly classified as ATTACK (True Positives), while 45.286 NORMAL flows were correctly classified as NORMAL (True Negatives). Classification errors consisted of 745 False Positives and 1.279 False Negatives. The precision value of 98,53% indicates that the majority of traffic predicted as ATTACK is indeed attack traffic. Meanwhile, the recall value of 97,50% indicates that the model successfully detected most of the DDoS attacks present in the test data. Although there were 1.279 attack traffic flows misclassified as NORMAL, this recall value still demonstrates a high level of detection capability. False Positives indicate the presence of NORMAL traffic that was incorrectly identified as an ATTACK, while False Negatives indicate the presence of DDoS traffic that the model failed to detect.

Analysis of Adaptation to Traffic Variations

An adaptation analysis was conducted to evaluate whether the AOS-ELM model maintained its classification performance when the characteristics and sequence of incoming attack traffic changed. The analysis used three controlled scenarios: attacks following the primary pattern, attacks with modified patterns, and distributed denial-of-service (DDoS) attacks. Each scenario was compared using accuracy, precision, recall, F1-score, and the classification errors observed in the confusion matrix. The comparison was intended to assess the sensitivity of the model to the evaluated changes in attack intensity, attack pattern, and traffic source distribution, as well as its ability to distinguish between NORMAL and ATTACK flow conditions under these different traffic scenarios. Differences in performance between scenarios were interpreted as evidence of different model responses to the tested traffic conditions and as an indication of practical adaptation under the laboratory testbed, rather than as formal proof of concept drift. Formal verification of concept drift would require an additional statistical analysis of feature-distribution changes or the use of an independent drift-detection method, which was outside the scope of this study. The results of the comparison are presented in Table 9.

Table 9 Comparison of Results from 3 Attack Pattern Tests

Test Parameters	Major Attack Patterns	Different Attack Patterns	DDoS Attacks
True Positive (TP)	48.497 Data Flow	46.598 Data Flow	49.922 Data Flow
True Negative (TN)	46.639 Data Flow	51.466 Data Flow	45.286 Data Flow
False Positive (FP)	218 Data Flow	3.964 Data Flow	745 Data Flow
False Negative (FN)	1.391 Data Flow	387 Data Flow	1.279 Data Flow
Akurasi (Accuracy)	98,34%	95,75%	97,92%
Presisi (Precision)	99,55%	92,16%	98,53%
Recall (Sensitivity)	97,21%	99,18%	97,50%
F1-Score	98,37%	95,54%	98,01%

Figure 4 visualizes the results in Table 9 and compares AOS-ELM performance across the three attack scenarios. The Major Attack Patterns scenario achieves the best overall performance, the Different Attack Patterns scenario achieves the highest recall, and the DDoS scenario maintains consistently high performance, with an F1-score of 98,01%.

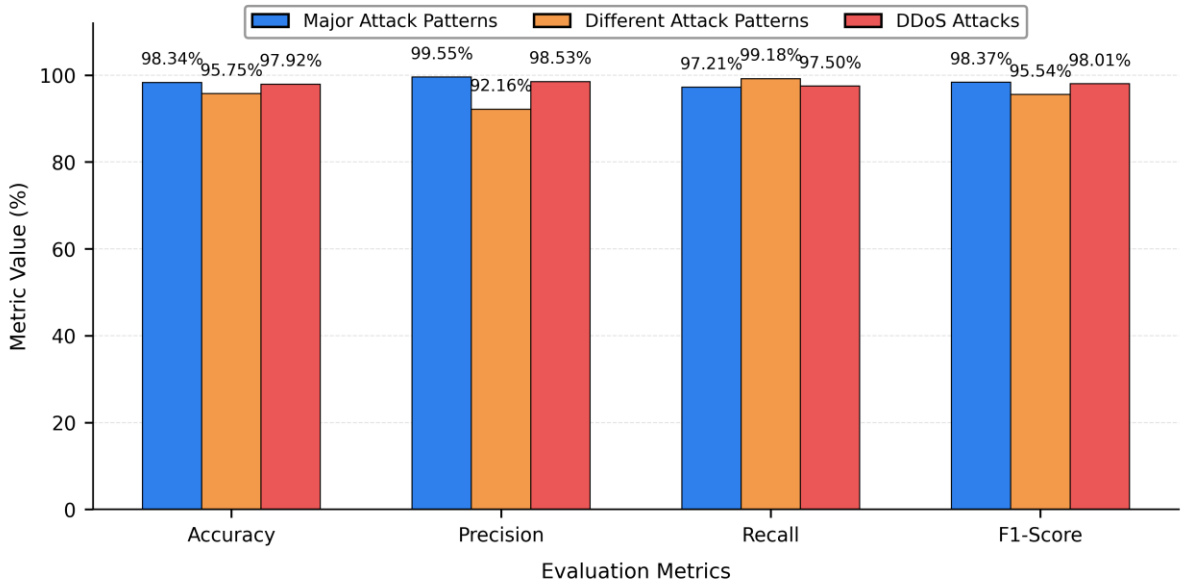

Figure 4 Comparison Chart of the 3 Attack Simulation Scenarios

Based on the comparison graph of the IDS model's performance across the three attack scenarios, the Major Attack Patterns scenario produces the best overall performance, particularly in terms of accuracy, precision, and F1-score. In this scenario, the model achieves an accuracy of 98,34%, a precision of 99,55%, a recall of 97,21%, and an F1-score of 98,37%. These results indicate that the model is highly effective at distinguishing between normal traffic and attack traffic when the attack patterns are consistent with the primary patterns recognized during the learning process. In the Different Attack Patterns scenario, the model's performance declines, particularly in terms of precision, which decreases to 92,16%. The accuracy also decreases to 95,75%, while the F1-score reaches 95,54%. However, recall remains very high at 99,18%. This indicates that the model is still highly effective in detecting attack traffic, but it produces more false-positive classifications, in which normal traffic is incorrectly classified as attack traffic. These results suggest that changes in attack patterns affect the model's ability to distinguish accurately between the NORMAL and ATTACK classes.

Meanwhile, in the DDoS Attacks scenario, the model achieves an accuracy of 97,92%, a precision of 98,53%, a recall of 97,50%, and an F1-score of 98,01%. These results demonstrate that the model is capable of handling DDoS traffic effectively, with performance that is slightly lower than that observed in the Major Attack Patterns scenario but higher than that of the Different Attack Patterns scenario in terms of accuracy, precision, and F1-score. Overall, the

three scenarios show that the model maintains strong attack-detection performance. Nevertheless, the best overall results are obtained when the attack patterns are similar to the primary patterns learned by the model. The high recall of 99,18% in the Different Attack Patterns scenario also indicates that the model retains strong sensitivity to previously unseen or modified attack behavior, although this comes with an increase in false-positive detections.

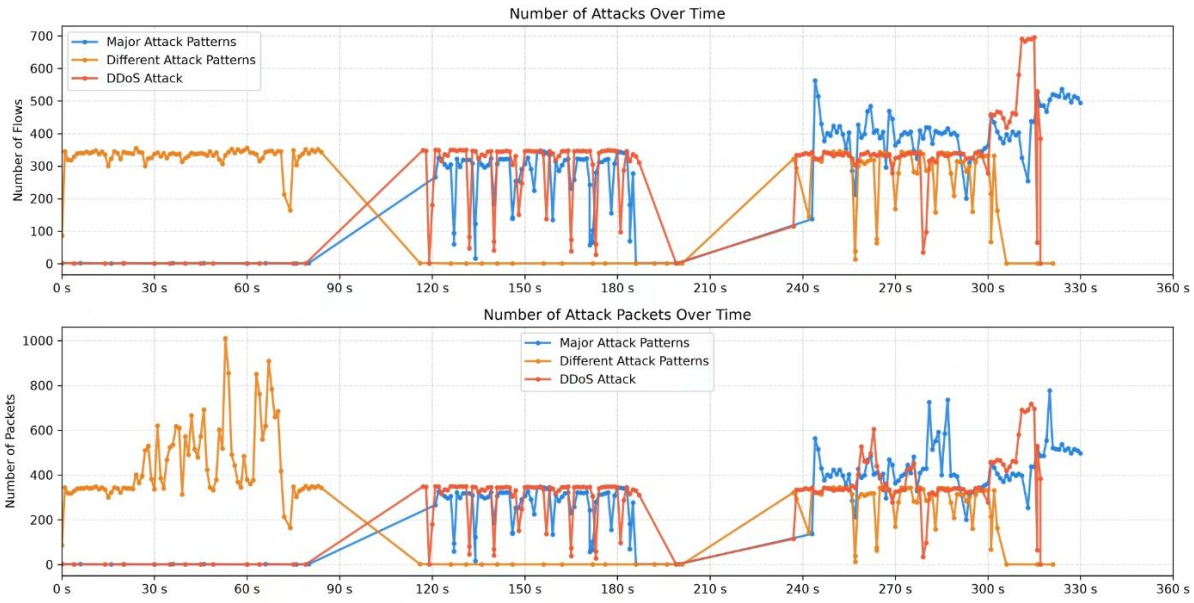

Figure 5 Attack Pattern Comparison

Based on the flow-count and packet-count graphs over time, the three testing scenarios exhibit distinct temporal characteristics. In the Different Attack Patterns scenario, the attack-flow count is relatively higher at the beginning of the observation period, decreases during the intermediate phase, and changes again toward the end of the test. This fluctuation indicates that the sequence and intensity of attack traffic differ from the main attack pattern. In the DDoS Attack scenario, the number of attack flows and packets shows a different distribution, with relatively stable activity during part of the observation period followed by more pronounced changes near the end. Meanwhile, the Major Attack Patterns scenario presents its own temporal pattern, reflecting attack behavior that is more consistent with the patterns used during model learning. These variations demonstrate that the volume, sequence, and intensity of attack traffic are not identical across the three scenarios. Such changes can be interpreted as a controlled distribution shift in streaming network traffic. Specifically, the transition from the Major Attack Patterns scenario to Different Attack Patterns and DDoS Attack scenarios represents changes in the temporal ordering and characteristics of attack traffic, which are relevant to the concept-drift setting. The change does not necessarily indicate that the model fails to recognize attacks; rather, it tests whether the model can maintain detection performance when incoming traffic differs from the patterns encountered during training.

Overall, the AOS-ELM-based IDS maintains strong detection performance across the three scenarios. The model achieves F1-scores of 98,37% for Major Attack Patterns, 95,54% for Different Attack Patterns, and 98,01% for DDoS Attacks. Recall remains high in all scenarios, reaching 97,21%, 99,18%, and 97,50%, respectively. The highest recall is obtained in the Different Attack Patterns scenario, indicating that the model remains highly sensitive to attack traffic even when the pattern differs from the primary attack sequence. However, its lower precision of 92,16% in this scenario suggests an increase in false-positive detections. Therefore, the results indicate that the AOS-ELM classifier remains effective under changes in attack sequence and traffic intensity, although pattern variation may reduce precision because normal traffic is more likely to be classified as an attack.

Conclusions

This study demonstrates that an AOS-ELM-based intrusion detection system can be deployed as a lightweight edge-based solution for monitoring Native API traffic between ESPHome devices and Home Assistant. The proposed framework integrates fixed time-window flow aggregation, statistical feature extraction, chronological data processing, sequential learning, and a bounded forgetting-factor mechanism to classify NORMAL and ATTACK traffic without requiring complete retraining using the entire historical dataset. The experimental findings show that the proposed IDS maintains effective and balanced detection performance under normal traffic, simulated DoS traffic, modified attack patterns, and distributed attack conditions, while remaining feasible for deployment on the evaluated edge devices. The results further indicate that hardware capability influences the balance among detection sensitivity, false alarms, and inference efficiency, with Raspberry Pi 5 providing the most suitable overall performance under the reported experimental conditions. The scientific contribution of this work is the development and empirical evaluation of a lightweight AOS-ELM IDS framework that combines adaptive sequential learning, flow-level Native API traffic analysis, and edge-computing deployment for ESPHome–Home Assistant IoT infrastructure. The present evaluation is limited to a controlled laboratory topology, Native API traffic, and simulated SYN Flood, Slowloris, modified DoS, and basic DDoS attack scenarios; therefore, future work should validate the framework in larger and more heterogeneous IoT deployments, evaluate additional protocols and attack types, compare alternative aggregation-window strategies, use independent statistical drift-detection methods, and extend the passive IDS toward carefully controlled automated response mechanisms.

References

Ahsan, M. S., Islam, S., & Shatabda, S. (2025). A systematic review of metaheuristics-based and machine learning-driven intrusion detection systems in IoT. *Swarm and Evolutionary Computation*, *96*. https://doi.org/10.1016/j.swevo.2025.101984

Alrawi, O., Lever, C., Antonakakis, M., & Monrose, F. (2019). SoK: Security Evaluation of Home-Based IoT Deployments. *Proceedings - IEEE Symposium on Security and Privacy*, *2019-May*, 1362–1380. https://doi.org/10.1109/SP.2019.00013

Alsubaei, F. S. (2025). Smart deep learning model for enhanced IoT intrusion detection. *Scientific Reports*, *15*(1). https://doi.org/10.1038/s41598-025-06363-5

Andrei-Traian, C., & Bogdan Felician, A. (2022). A Cyber Security Application For Reporting Malicious Attacks On Iot Networks That Use The Mqtt Protocol. *Industrial Engineering Journal*, *06*(03), 19–24.

Budiman, A., Fanany, M. I., & Basaruddin, C. (2016). Adaptive Online Sequential ELM for Concept Drift Tackling. *Computational Intelligence and Neuroscience*, *2016*. https://doi.org/10.1155/2016/8091267

Elsayed, N., Dzamesi, L., ElSayed, Z., & Ozer, M. (2025). *Extreme Learning Machine Based System for DDoS Attacks Detections on IoMT Devices*. http://arxiv.org/abs/2507.05132

Fatima, M., Rehman, O., Rahman, I. M. H., Ajmal, A., & Park, S. J. (2024). Towards Ensemble Feature Selection for Lightweight Intrusion Detection in Resource-Constrained IoT Devices. *Future Internet*, *16*(10). https://doi.org/10.3390/fi16100368

Garalov, T., & Elhajj, M. (2023). Enhancing IoT Security: Design and Evaluation of a Raspberry Pi-Based Intrusion Detection System. *2023 International Symposium on Networks, Computers and Communications, ISNCC 2023*, 1–7. https://doi.org/10.1109/ISNCC58260.2023.10323656

Guo, W. (2019). Robust adaptive online sequential extreme learning machine for predicting nonstationary data streams with outliers. *Journal of Algorithms and Computational Technology*, *13*. https://doi.org/10.1177/1748302619895421

Guo, W., Xu, T., Tang, K., Yu, J., & Chen, S. (2018). Online Sequential Extreme Learning Machine with Generalized Regularization and Adaptive Forgetting Factor for Time-Varying System Prediction. *Mathematical Problems in Engineering*, *2018*, 22. https://doi.org/10.1155/2018/6195387

Haider, A., Khan, M. A., Rehman, A., Ur Rahman, M., & Kim, H. S. (2020). A real-time sequential deep extreme learning machine cybersecurity intrusion detection system. *Computers, Materials and Continua*, *66*(2), 1785–1798. https://doi.org/10.32604/cmc.2020.013910

Hallizah, N., Dewi, I. K., Fernandes, A. L., & Saro, D. (2024). Improvement of IoT Security with a ML Based IDS Approach. *JR RESPONSIVE*, *8*(02), 105–113. https://doi.org/10.36352/jr.v8i02

Huang, G., Huang, G. Bin, Song, S., & You, K. (2015). Trends in extreme learning machines: A review. *Neural Networks*, *61*, 32–48. https://doi.org/10.1016/j.neunet.2014.10.001

Kumar, K. R. (2025). Intrusion Detection System using Raspberry PI for IoT Devices. *International Journal for Research in Applied Science and Engineering Technology*, *13*(4), 6496–6503. https://doi.org/10.22214/ijraset.2025.69909

Li, Y., Qiu, R., & Jing, S. (2018). Intrusion detection system using Online Sequence Extreme Learning Machine (OSELM) in advanced metering infrastructure of smart grid. *PLoS ONE*, *13*(2). https://doi.org/10.1371/journal.pone.0192216

Mahmud, M. Z., Islam, S., Alve, S. R., & Pial, A. J. (2024). Optimized IoT Intrusion Detection using Machine Learning Technique. *2024 IEEE 3rd International Conference on Robotics, Automation, Artificial-Intelligence and Internet-of-Things, RAAICON 2024 - Proceedings*, 167–172. https://doi.org/10.1109/RAAICON64172.2024.10928532

Pandey, V. K., Sahu, D., Prakash, S., Rathore, R. S., Dixit, P., & Hunko, I. (2025). A lightweight framework to secure IoT devices with limited resources in cloud environments. *Scientific Reports*, *15*(1). https://doi.org/10.1038/s41598-025-09885-0

Raje, V. V., Goel, S., Patil, S. V., Kokate, M. D., Mane, D. A., & Lavate, S. (2023). Realtime Anomaly Detection in Healthcare IoT: A Machine Learning-Driven Security Framework. *Journal of Electrical Systems*, *19*(3), 192–202. https://doi.org/10.52783/jes.700

Sai, K. M., Gupta, B. B., & Hsu, C. H. (2021). Lightweight Intrusion Detection System In IoT Networks Using Raspberry pi 3b+. *CEUR Workshop Proceedings*, *3080*.

Sallay, H. (2024). Designing an Adaptive Effective Intrusion Detection System for Smart Home IoT A Device-Specific Approach with SDN Deployment. *International Journal of Advanced Computer Science and Applications*, *15*(1), 937–948. https://doi.org/10.14569/IJACSA.2024.0150194

Saraladeve, L., Chandrasekar, A., Nithya, T., Mohamed Imtiaz, N., Kalaiarasi, S., Sampathkumar, B., Thanikachalam, R., & Maria Arockia Dass, J. (2025). A Multiclass Attack Classification Framework for IoT Using Hybrid Deep Learning Model. *Journal of Cybersecurity and Information Management*, *15*(1), 151–165. https://doi.org/10.54216/JCIM.150112

Wang, J., Lu, S., Wang, S., & Zhang, Y. (2022). A review on extreme learning machine JianTELLIGENCE. *Multimedia Tools and Applications*, *81*, 41611–41660.

Wibawa, I. P. D., Machbub, C., Rohman, A. S., & Hidayat, E. (2023). Modified online sequential extreme learning machine algorithm using model predictive control

approach. *Intelligent Systems with Applications*, *18*. https://doi.org/10.1016/j.iswa.2023.200191

Yang, L., & Shami, A. (2022). IDS-ML: An open source code for Intrusion Detection System development using Machine Learning[Formula presented]. *Software Impacts*, *14*, 100446. https://doi.org/10.1016/j.simpa.2022.100446

Zhang, X., & Yin, J. (2025). Optimized extreme learning machines with deep learning for high-performance network traffic classification. *Scientific Reports*, *15*(1), 1–15. https://doi.org/10.1038/s41598-025-16980-9