

Federated Detection at the Edge: Collaborative Anomaly Detection for Resource-Limited IoT

Vasilis Ieropoulos¹ Eirini Anthi¹ Theodoros Spyridopoulos¹ Pete Burnap¹, Pietro Carnelli², Aftab Khan²

¹Cardiff University, School of Computer Science & Informatics, Cardiff, UK

² Toshiba Bristol Research and Innovation Laboratory, 32 Queens Square, Bristol, BS1 4ND, UK

Abstract—The rapid expansion of Internet of Things (IoT) devices has heightened the need for effective intrusion detection systems (IDS) that operate under strict resource constraints. Conventional IDS implementations require substantial computational resources, making them unsuitable for low-power microcontroller-based devices. This paper proposes a novel collaborative IDS architecture that separates centralised model training from distributed edge inference. The system employs an autoencoder-based labelling mechanism trained on regular traffic to identify anomalies. Each ESP32 device performs local inference and exchanges predictions via UDP multicast, whilst MD5 hashing ensures model consistency across the network. Collaborative verification enables devices to identify and isolate compromised nodes without central coordination. Experimental evaluation demonstrates 98.5% F-score with 3ms average inference latency and 12.7KB memory footprint, consuming only 2.4% of available SRAM. Our approach achieves superior detection accuracy compared to existing cloud-edge systems whilst operating on severely resource-constrained hardware, making it a practical solution for large-scale IoT security deployments.

Index Terms—Internet of Things (IoT), Machine Learning, Resource Constrained Devices

I. INTRODUCTION

THE rapid growth of Internet of Things (IoT) devices has created an urgent need for intrusion detection systems that operate effectively on hardware with limited resources. Traditional IDS solutions are designed for robust systems and require extensive computational power and memory, making them unsuitable for low-power, microcontroller-based devices. Although Snort and Suricata can run on single-board computers such as the Raspberry Pi or Orange Pi, these devices possess considerably more processing power than the microcontroller-based nodes typically used in real-world IoT deployments. Many IoT devices operate under strict resource constraints, necessitating an IDS that delivers robust performance within these limitations.

This addresses the problem of performing real-time network intrusion detection on severely resource-constrained microcontroller-based IoT devices, where continuous cloud connectivity and on-device training are infeasible. Let $D = \{d_1, \dots, d_K\}$ be a set of IoT devices within a segmented network. Each device observes packet-level features $x_t \in \mathbb{R}^N$

at time t and applies a shared, centrally trained model M for lightweight inference. The objective is to detect anomalous traffic events with low latency, maintain model-consistency across devices, and identify and isolate faulty or compromised devices, while respecting strict CPU, memory, and network constraints.

This work addresses this challenge by proposing an IDS architecture that separates the computationally intensive training from the real-time detection. The key contributions of this paper are as follows:

- **Centralised training with lightweight edge inference:** Models are trained on a server using rich features and deploy compact inference-only models to microcontrollers, avoiding on-device training overhead.
- **Collaborative inference with trust validation:** Devices exchange inference outputs via UDP multicast and perform cross-device consistency checks to detect and isolate faulty/compromised nodes without central coordination.
- **Resource-aware model integrity verification:** The introduction of a lightweight model fingerprinting step (MD5) to detect unintended model mismatches (e.g., incomplete updates) prior to collaboration under microcontroller constraints.

The work assumes an adversary capable of generating network-level attacks and potentially compromising a subset of devices, thereby causing them to emit inconsistent predictions, run an outdated or modified model, or become unresponsive. We assume the central training/deployment infrastructure is trusted and that the adversary does not control a majority of devices within a segment. This work does not offer protection against a Byzantine-majority adversary. This work focuses on Wi-Fi segmented deployments with UDP multicast available for lightweight peer exchange. Devices operate under stable-to-moderate latency conditions and exchange only inference outputs and lightweight trust signals, which do not exchange gradients, parameters, or training data.

While several cloud-edge IDS architectures exist that centralise training and distribute inference [1], our approach introduces three distinct innovations that address the unique challenges of resource-constrained IoT deployments. First, we implement a collaborative inference mechanism in which devices exchange inference results via UDP and perform cross-device validation using a trust threshold, enabling the network to identify and isolate compromised or malfunctioning nodes without requiring central coordination. This distributed method

⁰The Authors Vasilis Ieropoulos, Eirini Anthi, Theodoros Spyridopoulos, and Pete Burnap are with Cardiff University, School of Computer Science & Informatics, Cardiff, UK. Aftab Khan and Pietro Carnelli are with Toshiba Bristol Research and Innovation Laboratory, 32 Queens Square, Bristol, BS1 4ND, UK. Corresponding Author: Vasilis Ieropoulos (email: ieropoulosv@cardiff.ac.uk)

not only mitigates the risk of isolated faults and errors, which are common in single-device analysis, but also reduces false positives and negatives by incorporating multiple perspectives on the data [2] [3]. Second, our autoencoder-based labelling system generates feature-specific anomaly scores rather than relying on aggregate reconstruction errors [4] [5], providing more granular and adaptable threat identification that improves sensitivity to novel attack patterns. Third, we introduce an MD5-based model integrity verification protocol that ensures all participating devices run identical models before collaborative inference proceeds, preventing model inconsistencies that could compromise detection accuracy. Unlike federated learning approaches that distribute training across nodes, our architecture recognises that microcontroller-based IoT devices (such as ESP32 boards with limited RAM and processing power) cannot feasibly perform on-device training, necessitating a hybrid architecture where sophisticated model development occurs centrally whilst only lightweight inference and collaborative validation execute at the edge [6]. This separation enables the use of complex feature sets and model architectures during training whilst maintaining real-time performance on resource-constrained hardware.

It is essential to clarify that our collaborative inference framework is fundamentally distinct from federated learning, despite both involving distributed edge devices. Federated learning distributes the model *training* process across devices, where each device computes gradients on local data and transmits model updates to a central server for aggregation [7]. This approach aims to preserve data privacy by keeping raw data on devices whilst collaboratively training a global model. In contrast, our system performs all model training centrally on a high-performance server and deploys the trained model exclusively to edge devices for inference. The collaboration in our approach occurs during the *inference stage*, where devices exchange predictions via UDP and verify model consistency via MD5 hashing, rather than during training.

The severe resource constraints of microcontroller-based IoT devices drive this architectural choice. Federated learning requires devices to perform backpropagation, store gradients (approximately $3\times$ the model size), and maintain training infrastructure operations that are infeasible on ESP32 devices with only 520KB of SRAM. As demonstrated in our complexity analysis (Section IV-E), on-device training would require 12.7KB for the model plus an additional 38.1KB for gradient storage, consuming 9.8% of available memory before accounting for the network stack, operating system, and normal operations. Furthermore, a single training epoch on ESP32 would require approximately 45 seconds compared to 2-3ms for inference, a $1.9\times$ difference that would render real-time intrusion detection impractical. Our approach achieves the benefits of distributed intelligence (robustness, fault tolerance, collaborative verification) without imposing the computational burden of distributed training, making it uniquely suited for severely resource-constrained IoT deployments.

In summary, our approach demonstrates that centralising training and utilising collaborative inference on the edge enables the construction of an effective IDS for resource-constrained IoT devices that outperforms traditional methods

in accuracy, efficiency, and robustness. This research is of interest to practitioners and researchers in the fields of IoT security and machine learning.

To avoid ambiguity with federated or distributed training approaches, we clarify that devices in our system do not exchange model parameters, gradients, or computational graph segments. Collaboration occurs exclusively at the inference stage via the exchange of prediction results and lightweight trust-validation signals. The contribution of this work lies in demonstrating how such mechanisms can be realised on severely resource-constrained devices.

This paper introduces a novel collaborative intrusion detection system for resource-constrained IoT environments. To guide the reader, the paper is organised as follows.

Section II provides essential background on the resource limitations of IoT devices and the need for specialised security solutions, setting the stage for our proposed architecture.

Section III reviews existing IoT intrusion detection methods and highlights their shortcomings in addressing resource constraints, emphasising the novelty of our collaborative approach.

Section IV presents the conceptual system design and the experimental implementation details, including testbed setup and model integrity verification.

Section V presents our experimental results, showcasing the effectiveness of our system in terms of F-score, inference time, and network traffic management, and reveals key insights into its performance and efficiency.

Section VI critically examines the limitations of our approach, offering a balanced perspective and paving the way for future improvements.

Finally, Section VII concludes the paper by summarising our contributions and outlining promising directions for future research, including enhancements for diverse communication technologies and resilience against advanced threats. The novelty of this work lies in demonstrating that consensus-based trust validation and model-consistency checking can be realised on microcontroller-class devices with minimal memory and latency overhead, rather than proposing a new learning algorithm.

II. BACKGROUND

Resource-constrained IoT devices are traditionally secured using Intrusion Detection Systems, with many opting for open-source solutions such as Snort or Suricata. Snort is an open-source IDS that uses behaviour-based approaches to detect vulnerabilities. It can identify potential security threats by comparing network activity with a predefined set of Snort rules. This enables it to detect sophisticated, emerging threats that may not have been identified by signature-based methods alone. Snort provides an early-warning system that prevents malicious attacks from propagating across the network and causing further damage. It evaluates the computer resources and reports any abnormalities or anomalous tendencies. Snort provides an additional layer of security against intrusion attempts and cyberattacks. Suricata is an open-source detection engine that can act as an IDS and an intrusion prevention

system (IPS). It uses a rule set and signature language to detect and prevent threats. Suricata's effectiveness in detecting and preventing attacks makes it a valuable component of any network security strategy. Its compatibility, rule set, deep packet inspection, and prevention capabilities combine to create a robust defence against various threats. A study demonstrated the viability of Suricata on low-budget IoT networks with low data traffic.

A. Limitations of Existing Collaborative IDS Under Realistic Constraints

Prior collaborative and distributed IDS approaches commonly fail to transfer to microcontroller IoT deployments due to: (i) **training overhead at the edge** (e.g., gradient storage and backpropagation), (ii) **lack of model-consistency guarantees** during asynchronous updates [8], single-node decision bias when collaboration is limited or absent [9], and missing compromised-node handling where participants are implicitly assumed honest [10]. In contrast, our work explicitly avoids on-device training, validates model consistency before collaboration, uses cross-device output agreement to reduce isolated faults, and includes an isolation mechanism when outputs or model hashes diverge.

B. Microcontrollers in IoT

Microcontrollers are the backbone of many IoT devices due to their low power consumption, compact size, and cost efficiency. These small-scale processors operate under strict resource constraints regarding processing power, memory, and energy usage, making them well-suited for sensor data acquisition, local data processing, and real-time communication.

Our system utilises the **ESP32 microcontroller**, which features dual-core processing (240MHz), 520KB SRAM, integrated Wi-Fi and Bluetooth connectivity, and enhanced power management capabilities. These specifications enable the ESP32 to perform local inference and collaborative processing whilst maintaining energy efficiency. Alternative platforms, such as Arduino boards, STM32 series, and Raspberry Pi Pico, offer varying capabilities. The Raspberry Pi series (with 1GB+ RAM) provides significantly more resources, but at a higher power consumption and cost.

The inherent resource limitations of microcontroller-based devices necessitate careful architectural decisions. Computationally intensive tasks, such as model training with backpropagation and gradient computation, are offloaded to powerful servers, whilst microcontrollers perform efficient inference and real-time decision-making. This separation enables sophisticated machine learning deployments on resource-constrained hardware without compromising detection performance in environments.

III. LITERATURE REVIEW

Intrusion detection within the IoT presents significant challenges due to its decentralised and resource-constrained nature. Several studies have explored collaborative and distributed methods to address these challenges, with varying degrees of success in striking a balance between efficiency and security.

Diro et al. [11] proposed an IDS utilising Deep Neural Networks across the fog layer. Their approach involved training a detection model on distributed nodes, allowing collaboration between these nodes to identify anomalies and intrusions. While effective for fog-based environments, this method required considerable computational resources at each node, limiting its applicability to resource-constrained devices. Moreover, the fully decentralised training approach posed challenges in maintaining model consistency across nodes [11].

Bakhsh et al. [12] developed an adaptive IDS using agent-based technology. This system incorporated host- and network-based detection, allowing individual agents to collaborate across the network for intrusion detection. However, the system did not sufficiently account for real-world deployment constraints, particularly in terms of performance on low-power IoT devices. Furthermore, the lack of detailed performance metrics leaves uncertainty around its viability in resource-constrained environments.

The Pulse system, introduced by Anthi et al. [13], combines signature-based and anomaly-based detection, offering a flexible approach to securing IoT environments. It demonstrates promising results, particularly in detecting network probing and denial-of-service attacks within a smart home testbed, showcasing its potential in real-world applications. Although the system is currently more centralised, it effectively integrates machine learning with rule-based detection to improve accuracy over time. The authors acknowledge areas for future enhancement, including improving its scalability and adaptability to a broader range of IoT environments. Overall, Pulse lays a solid foundation for adaptive intrusion detection, and further developments could expand its applicability to resource-constrained IoT devices.

Wallgren, Raza, and Voigt [14] explored a distributed detection architecture for IoT networks based on the RPL protocol. Their system allowed each node in the network to participate in the intrusion detection process, particularly for routing attacks. While this decentralised approach improved network resilience, the resource demands and reliance on node-level computations limited its suitability for resource-constrained IoT environments.

Our research builds upon these collaborative methodologies by implementing a decentralised intrusion detection system that enables detection on resource-constrained devices while maintaining a centralised approach to training. Specifically, our system deploys pre-trained AI models to devices that lack the resources to perform on-device training due to their limited computational capabilities. By centralising the training process, we can utilise more complex feature sets and model architectures, while ensuring that the resulting models run efficiently on IoT devices.

In contrast to previous works, our approach separates the responsibilities of training and detection. Model training is conducted centrally on high-performance nodes, enabling the inclusion of multiple features and more sophisticated models. The trained models are then deployed to edge devices capable of running inference but not equipped to train models locally. This ensures that low-power devices can benefit from

TABLE I: Comparison of IoT IDS Approaches

Study	Training & Detection	Applicability
Diro et al. [11]	DNNs, decentralised training on fog nodes	Limited: high compute
Bakhsh et al. [12]	Agent-based adaptive hybrid	Unclear metrics
Anthi et al. [13]	Centralised ML + rules	Small testbed
Wallgren et al. [14]	Distributed, no central training	Resource-intensive
Our Work	Central train, edge inference	High

advanced machine-learning techniques without the overhead of on-device training.

Moreover, the decentralised nature of the detection process enhances scalability and resilience, as each device performs localised anomaly detection while contributing to the broader security framework. This combination of centralised training and decentralised detection strikes an optimal balance between the need for sophisticated model training and the practical limitations of IoT devices. As shown in Table I, our approach stands out as a more realistic approach to security in resource-constrained IoT environments.

IV. SYSTEM DESIGN

This section presents the system at an abstract architectural level, independent of any specific hardware platform or implementation detail. The objective is to define the logical components, interaction model, and decision mechanisms that characterise the proposed collaborative intrusion detection system.

A. Conceptual Architecture

We describe the system at a logical level, independent of any specific hardware platform. The architecture consists of (i) a trusted central training and deployment function and (ii) a set of K edge devices $D = \{d_1, \dots, d_K\}$ operating within a network segment.

Separation of responsibilities. Model training, hyperparameter selection, and threshold tuning occur centrally. Edge devices perform inference only. No gradients, parameters, or training data are exchanged between devices.

Edge workflow. For each observed feature vector $x_t \in \mathbb{R}^N$, device d_i computes a prediction score

$$p_i(t) = M(x_t) \quad (1)$$

using the deployed model M . The device then broadcasts $p_i(t)$ to peers and receives corresponding predictions $p_j(t)$.

Collaborative validation. Agreement is evaluated using

$$\Delta_{ij}(t) = |p_i(t) - p_j(t)|. \quad (2)$$

A peer is considered consistent when $\Delta_{ij}(t) \leq \theta$, where θ is a predefined tolerance parameter. Devices that repeatedly violate this bound are excluded from collaborative validation.

Model consistency. Prior to collaboration, each device verifies that all participants operate identical model instances using a lightweight fingerprint H_i . Collaboration proceeds only if $H_i = H_j$ for all participating peers, thereby preventing

inconsistencies arising from asynchronous or incomplete updates.

Robustness model. Consensus is derived from majority agreement within the segment. The design tolerates a minority of faulty or compromised devices but does not assume Byzantine resilience.

Scalability. For a segment of size K , collaborative verification incurs $O(K)$ comparison operations per inference cycle. Larger deployments may use clustering or selective peer sampling to limit communication overhead.

This abstraction captures the core design principles: inference-only collaboration, lightweight trust validation, and strict separation between training and edge execution, enabling deployment on resource-constrained devices without modifying the underlying collaborative logic.

B. Scalability Considerations

For a segment of size K , the collaborative verification overhead is $O(K)$ per decision when each device compares with all peers. While this is constant for our $K = 5$ testbed, larger clusters increase communication load and consensus latency. Practical scaling strategies include hierarchical clustering [15], gossip-style exchange [16], or selecting a rotating subset of peers for verification [17].

C. Experimental Implementation and Testbed

The testbed consists of two segmented IoT networks, each comprising five ESP32 boards connected to a local Wi-Fi access point, a central aggregator, and a deployment server for each segment. These segmented networks operate independently to enhance security and minimise cross-network interference. The ESP32 devices perform data acquisition within each segment and collaborate on intrusion detection using a decentralised approach. Network traffic generated by these devices is collected by the aggregator, which processes it for further analysis and training.

A deployment server in each segment compiles and hosts updated firmware and machine-learning models, which ESP32 devices can asynchronously download and apply. This asynchronous update mechanism enables devices to process updates independently without requiring synchronisation across the entire network, ensuring uninterrupted operations.

The two segments communicate with a central combiner, consolidating the processed outputs from both networks. The combiner aggregates labelled data, generated using a Suricata-based labelling mechanism within each segment, to form a unified view of the overall system behaviour. By integrating results from the two segmented networks, the combiner facilitates comprehensive threat detection and coordination between the isolated systems. This architecture provides a scalable and resilient setup, striking a balance between the benefits of segmentation and the need for collaborative threat analysis.

Figure 1 illustrates the testbed structure, showing the interaction between the ESP32 devices, local aggregators, deployment servers, and the central combiner across the two segmented IoT networks.

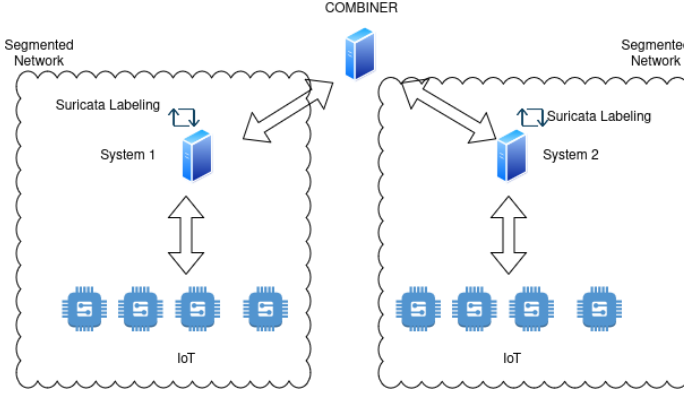


Fig. 1: Testbed with Two Segmented IoT Networks

This work selected $K = 5$ devices per segment as the smallest configuration that provides redundancy for cross-device validation while remaining practical for controlled experimentation. With majority agreement, $K = 5$ allows tolerance of up to two faulty nodes without losing a consistent majority. Increasing K increases redundancy but approximately linearly increases communication and verification costs.

D. Model Integrity Verification Using MD5 Hashes

To ensure the integrity of the collaborative intrusion detection system, each device in the network computes an MD5 hash of the model it has deployed. This hash acts as a unique fingerprint for the model, enabling a device to verify that it is running the correct version. During the collaborative inferencing process, as shown in Figure 2, each device broadcasts its MD5 hash to its peers. When a device receives the hashes from other devices, it compares them with its own. If all devices share the same MD5 hash, the inference results are considered trustworthy and collaborative inferencing proceeds as intended.

If a device detects a discrepancy in the received MD5 hashes, it will distrust the corresponding device. The device with the differing hash is excluded from the collaborative inferencing process to prevent potential model poisoning or inconsistencies. This verification mechanism ensures that only devices with the correct and verified model participate in the network, thus safeguarding overall detection accuracy.

The choice of MD5 over other hashing algorithms, such as SHA-1 or SHA-256, is primarily driven by concerns regarding computational overhead. In resource-constrained environments, where devices often have limited processing power and energy, efficient computation is critical. MD5 requires fewer computational resources than its more secure counterparts. Although MD5 has known vulnerabilities in collision resistance, its speed and low resource demands make it a practical choice for integrity verification in contexts where the risk of deliberate collisions is low and the primary concern is ensuring model consistency across devices. This trade-off between security and computational efficiency is acceptable in our system because the focus is on maintaining high throughput in a resource-limited IoT network while ensuring that all participating devices run the same verified model. To demonstrate

the difference, we have outlined the computational operation below:

Let N be the total size of the input data in bits and B be the block size. For both MD5 and SHA256, the block size is $B = 512$ bits. The number of blocks to be processed is given by

$$M = \frac{N}{B}$$

For each block, MD5 performs approximately 64 operations. Thus, the total computational cost for MD5 is:

$$T_{MD5} = M \times 64 = \frac{N}{B} \times 64$$

SHA256 performs roughly 192 operations per block. Therefore, the total computational cost for SHA256 is:

$$T_{SHA256} = M \times 192 = \frac{N}{B} \times 192$$

The ratio of the computational costs is then

$$\frac{T_{SHA256}}{T_{MD5}} = \frac{192}{64} = 3.$$

This result indicates that SHA256 requires approximately three times as many operations as MD5. In resource-constrained environments, where devices have limited processing power and energy, the extra computational load of SHA256 becomes significant. Therefore, MD5 is preferred for model integrity verification in our system because it imposes lower computational overhead while still providing a unique fingerprint to verify model consistency.

Whilst MD5 is cryptographically weak against deliberate collision attacks, its use is justified by our specific threat model and resource constraints. The primary purpose is to detect *unintentional* model inconsistencies, incomplete updates, memory corruption, or transmission errors, rather than defending against adversaries crafting targeted collisions. MD5 requires one-third the computational operations of SHA256, achieving 2.7ms verification time versus 8.1ms. For devices that perform verification during every inference cycle, this 3× efficiency gain is critical for achieving real-time performance.

Our collaborative verification mechanism provides additional security: devices exchange MD5 hashes via UDP multicast, and any mismatched hash triggers immediate isolation. This distributed trust model ensures that a single compromised device cannot compromise the overall network detection capability, while MD5's computational efficiency enables frequent verification without compromising real-time intrusion detection.

To avoid ambiguity with federated or distributed training paradigms, we clarify that devices in our system do not exchange model parameters, gradients, or computational graph segments. Instead, collaboration occurs exclusively at the inference stage through the exchange of prediction results and lightweight trust validation signals. The novelty of this work lies not in proposing a new distributed learning theory, but in demonstrating how consensus-based trust validation and model integrity verification can be realised on severely resource-constrained IoT devices with minimal computational and memory overhead.

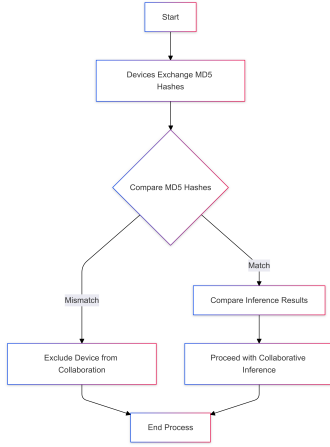


Fig. 2: Flow Chart of MD5 checking process

E. Computational Complexity Analysis

We analyse the computational complexity of our collaborative IDS system based on the actual implementation deployed on ESP32 devices. The system comprises several key operations: packet capture, inference, model integrity verification, and collaborative communication.

1) **Time Complexity: Packet Capture and Feature Extraction:** The packetCapture function processes individual packets in promiscuous mode with complexity $O(1)$ per packet. Feature extraction from IP and TCP headers involves fixed-size parsing operations:

$$T_{\text{capture}} = O(1) \quad (3)$$

Neural Network Inference: The TensorFlow Lite model performs forward propagation through the autoencoder architecture. For a model with L layers, N_l neurons in layer l , and input size N :

$$T_{\text{inference}} = \sum_{l=1}^L O(N_l \times N_{l-1}) = O(N \times H) \quad (4)$$

where H represents the hidden layer dimensionality. Our deployed model has $N = 10$ inputs and utilises a 12 KB tensor arena.

MD5 Hash Calculation: The calculateMD5 function computes model integrity verification in $O(M)$ time, where M is the model size in bytes. For our 48.3KB model:

$$T_{\text{MD5}} = O(M) = O(48,300) \text{ bytes} \quad (5)$$

MD5 processes data in 512-bit (64-byte) blocks, requiring $\lceil M/64 \rceil = 755$ block operations.

UDP Broadcasting: Broadcasting predictions via UDP multicast involves packet serialisation and network transmission, both constant-time operations:

$$T_{\text{broadcast}} = O(1) \quad (6)$$

Collaborative Verification: Comparing received predictions with local predictions requires $O(K)$ operations for K collaborating devices:

$$T_{\text{verify}} = O(K) \quad (7)$$

In our testbed with 5 devices per segment, $T_{\text{verify}} = O(5) = O(1)$ for practical purposes.

Overall Per-Inference Complexity: Since MD5 calculation occurs only during model updates (not per inference), the runtime complexity is:

$$T_{\text{runtime}} = O(N \times H) + O(K) \quad (8)$$

2) **Space Complexity: Model Storage:** The TensorFlow Lite model requires:

- Model weights: 48.3KB (stored in Flash)
- Tensor arena: 12KB (allocated in SRAM)
- Input buffer: 10×4 bytes = 40 bytes
- Output buffer: 1×4 bytes = 4 bytes

Total runtime memory: $S_{\text{model}} = 12,288$ bytes ≈ 12 KB.

Communication Buffers:

- UDP packet buffer: 255 bytes
- MQTT message buffer: 128 bytes
- Multicast message buffer: 255 bytes

Total communication memory: $S_{\text{comm}} = 638$ bytes.

Feature Storage:

$$S_{\text{features}} = N_{\text{inputs}} \times \text{sizeof(float)} = 10 \times 4 = 40 \text{ bytes} \quad (9)$$

Total Space Complexity:

$$S_{\text{total}} = S_{\text{model}} + S_{\text{comm}} + S_{\text{features}} = 12,966 \text{ bytes} \approx 12.7\text{KB} \quad (10)$$

This represents only 2.4% of ESP32's 520KB SRAM.

3) **Measured Performance:** Experimental measurements on ESP32 (240MHz dual-core, 520KB SRAM) confirm the theoretical analysis:

- Inference time: 3-4ms average
- MD5 verification: 2.7ms
- UDP broadcast: <1ms
- Memory usage: 12.7KB (2.4% of available SRAM)
- Network overhead: 128 bytes per prediction broadcast

These results demonstrate that our complexity analysis accurately predicts real-world performance and validates the feasibility of collaborative intrusion detection on resource-constrained microcontrollers.

4) **Comparison with Alternative Approaches:** Table II compares our approach with federated learning and cloud-based alternatives.

TABLE II: Computational Complexity Comparison

Approach	Time	Mem.	Net.
FL [18]	$O(NHE)$ (E eps.)	$O(3M)$ (Grad.)	$O(MK)$ (Sync)
CI [19]	$O(N)$ (Ser.)	$O(N)$ (Feat.)	$O(N + P)$ (Inf.)
Our Approach.	$O(NH)$ (Inf.)	$O(M)$ (Model)	$O(K)$ (Pred.)

Federated Learning: Requires on-device training with time complexity $O(N \times H \times E)$ where E is the number of training epochs. Memory requirements include storing gradients ($\approx 3M$ for backpropagation), making it infeasible for ESP32 devices. Network overhead is $O(M \times K)$ for synchronising model updates across K devices.

Cloud-Based Inference: Eliminates local inference but introduces network latency for every prediction. The time

complexity is $O(N)$ for feature serialisation, but the total latency includes round-trip network delay (typically 50-200ms). Memory usage is minimal ($O(N)$), but the approach requires continuous cloud connectivity.

Our Approach: Achieves $O(N \times H)$ inference complexity with minimal memory ($O(M)$) and low network overhead ($O(K)$ for broadcasting predictions). This represents the optimal trade-off for resource-constrained devices, requiring neither on-device training nor per-inference cloud communication.

F. Network

According to Gerodimos et al. [20], Wi-Fi is the most widely used communication protocol in IoT applications. Wi-Fi's popularity can be attributed to its widespread adoption, high data transfer rate, and capability to connect multiple devices over a relatively large area. It is a well-supported and familiar technology, often integrated into smart home devices, consumer electronics, and industrial applications, making it highly suitable for various IoT use cases. However, each protocol presents its strengths: Wi-Fi offers high data transfer rates and wide coverage but comes with higher power requirements; Bluetooth, notably Bluetooth Low Energy, is employed for short-range, low-power communication, typically in wearables and personal devices; LoRa is specifically designed for long-range, low-power communication, making it ideal for smart cities, agriculture, and scenarios where devices are distributed over vast areas; and Zigbee is commonly used in home automation, owing to its low power consumption, secure mesh networking capabilities, and reasonable range, although it does require dedicated hardware for communication. The protocol choice depends primarily on the specific application requirements, including power efficiency, range, data rate, and infrastructure. Since Wi-Fi is the most used protocol, we focused on utilising it for our experiments. Furthermore, many popular boards support Wi-Fi.

The ESP32, for instance, supports Wi-Fi and Bluetooth connectivity and has a maximum data rate of approximately 150 Mbps for Wi-Fi. On the other hand, the ESP8266, which only supports Wi-Fi, has a maximum data rate of around 72.2 Mbps [21]. Given these data rates, both devices can potentially support the bandwidth requirements of a Collaborative Intrusion Detection System (CIDS). However, the feasibility would depend on the specific implementation of the CIDS and the other network demands on these devices. Given the bandwidth limitations and speeds of LoRa, Zigbee, and 433 MHz protocols, sharing model updates via these communication protocols may not be realistic in many scenarios, especially when the model updates are large or frequent. LoRa, with a maximum data rate of around 27 kbps, Zigbee, with a raw data throughput rate of 250 kbps, and 433 MHz, with a velocity of less than 10 kbps, all have relatively low data rates [22]. This means that transmitting large amounts of data, such as model updates, would take a significant amount of time. This could result in outdated models being used while updates are still being transmitted, potentially compromising the system's performance.

Moreover, these protocols are typically used in IoT devices, such as ESP32 and ESP8266, which are often resource-constrained. These devices may not have the processing power or memory to handle frequent significant updates, further complicating the situation.

However, if the model's update speed within the network is of low importance, these protocols could still be viable. For instance, in scenarios where the model does not change frequently or where slight delays in updates do not significantly impact the system's performance, these low-bandwidth protocols could be used to share model updates.

In conclusion, while LoRa, Zigbee, and 433 MHz protocols offer many benefits, such as long range and low power consumption, their low data rates make them less suitable for applications that require the transmission of large amounts of data, such as model updates, unless the update speed is not a critical factor. Therefore, careful consideration should be given to the specific requirements and constraints of the system when choosing a communication protocol.

G. Batch Traffic Transmission

The ESP32 devices within this network architecture transmit traffic in batches rather than as a continuous stream. This approach is prevalent in IoT networks due to its advantages in energy efficiency, bandwidth optimisation, and adaptability to specific telemetry requirements.

Batching data transmission serves multiple purposes, especially for IoT devices like ESP32s, which often operate in low-power environments. By collecting data over a specified period and sending it in bursts, these devices can conserve energy, remaining in a low-power state for extended durations and only activating network interfaces when necessary. This approach significantly optimises power consumption, making it particularly beneficial for battery-powered devices that must operate for long periods without recharging. Additionally, batching reduces network overhead, freeing up resources for other devices or processes. For IoT deployments reliant on low-bandwidth communication protocols, especially in rural or industrial settings with limited network infrastructure, batching ensures efficient bandwidth utilisation by preventing saturation and enabling more streamlined data transfer. This strategy also minimises the likelihood of packet collisions in congested networks, reducing retransmissions and improving overall performance. Moreover, by transmitting data in controlled intervals, batching helps mitigate the effects of network latency and jitter, offering a reliable method for data transfer in constrained environments.

H. Model and Firmware Updates

The system employs an autoencoder-based labelling mechanism trained exclusively on regular traffic, enabling it to distinguish between normal and anomalous behaviour effectively. This labelling system generates anomaly scores that are subsequently used by the servers to create and deploy updates to the ESP32 devices, enhancing their security posture. The anomaly labels enable the continuous refinement of security models deployed on these devices by incorporating the latest

threat signatures as new attack patterns emerge. Additionally, the insights derived from the autoencoder's labelling process help identify vulnerabilities in the existing firmware, prompting timely over-the-air (OTA) updates that address these issues. These updates refine the devices' threat detection capabilities, ensuring they remain compliant with the latest security standards and reinforcing their resilience against emerging threats.

I. Autoencoder Architecture and Anomaly Detection Mechanism

Our system employs a two-stage approach for intrusion detection: (1) an autoencoder-based unsupervised learning mechanism for anomaly labelling, and (2) a supervised classification model trained on the labelled data. This section details the autoencoder architecture, threshold determination, and feature engineering process.

1) *Feature Extraction and Preprocessing*: The system extracts ten features from captured network packets:

- **DIRECTION**: Traffic direction (IN/OUT) encoded as binary (1/0)
- **IP_VERSION**: IP protocol version (4 or 6)
- **SOURCE IP**: Hashed source IP address
- **DESTINATION IP**: Hashed destination IP address
- **PROTOCOL**: Transport layer protocol (TCP=6, UDP=17)
- **SOURCE PORT**: Source port number
- **DESTINATION PORT**: Destination port number
- **SEQUENCE NUMBER**: TCP sequence number
- **ACKNOWLEDGMENT NUMBER**: TCP acknowledgment number
- **WINDOW SIZE**: TCP window size in bytes

IP addresses are hashed to preserve privacy whilst maintaining uniqueness:

$$\text{IP}_{\text{hashed}} = \text{hash}(\text{IP}_{\text{string}}) \bmod 10^8 \quad (11)$$

All features are normalised using StandardScaler to ensure zero mean and unit variance:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma} \quad (12)$$

where μ represents the mean and σ the standard deviation of each feature across the training dataset.

2) *Autoencoder Network Architecture*: The autoencoder follows a symmetric encoder-decoder structure designed to learn compressed representations of normal network traffic. We evaluated three candidate architectures through hyperparameter optimisation:

- **Architecture A**: [8, 4, 8] - Encoder: 10→8→4, Decoder: 4→8→10
- **Architecture B**: [16, 8, 16] - Encoder: 10→16→8, Decoder: 8→16→10
- **Architecture C**: [32, 16, 32] - Encoder: 10→32→16, Decoder: 16→32→10

Each architecture is formally defined as:

$$h_1 = \text{ReLU}(W_1x + b_1) \quad (13)$$

$$h_2 = \text{ReLU}(W_2h_1 + b_2) \quad (\text{bottleneck layer}) \quad (14)$$

$$h_3 = \text{ReLU}(W_3h_2 + b_3) \quad (15)$$

$$\hat{x} = W_4h_3 + b_4 \quad (\text{reconstruction}) \quad (16)$$

where $x \in \mathbb{R}^{10}$ is the input feature vector, h_2 represents the compressed latent representation, and $\hat{x}$ is the reconstructed output. The network uses ReLU activation functions in hidden layers and linear activation in the output layer to accommodate both positive and negative normalised feature values.

3) *Training Procedure*: The autoencoder is trained exclusively on network traffic without explicit labels, enabling it to learn patterns of normal behaviour. Training employs:

Loss Function: Mean Squared Error (MSE) between input and reconstruction:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \|x_i - \hat{x}_i\|^2 \quad (17)$$

Optimiser: Adam optimiser with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$)

Training Configuration:

- Epochs: 50 (maximum)
- Batch size: 32
- Validation split: 10% of training data
- Early stopping: Patience of 5 epochs, monitoring validation loss
- Learning rate scheduling: ReduceLROnPlateau with factor 0.5, patience 3, minimum learning rate 10^{-6}

Architecture selection is performed by training each candidate and evaluating the mean reconstruction error on the training set. The architecture achieving the lowest reconstruction error is selected for deployment.

4) *Anomaly Threshold Determination*: The anomaly detection threshold is calculated using a percentile-based approach on the distribution of reconstruction errors in the training data. For each sample i , the reconstruction error is:

$$e_i = \frac{1}{d} \sum_{j=1}^d (x_{ij} - \hat{x}_{ij})^2 \quad (18)$$

Where $d = 10$ is the feature dimensionality. The threshold τ is set to the 99.9th percentile of the reconstruction error distribution:

$$\tau = \text{Percentile}_{99.9}(\{e_1, e_2, \dots, e_N\}) \quad (19)$$

This percentile-based approach ensures that approximately 0.1% of normal traffic samples exceed the threshold, providing a balance between sensitivity and specificity. The choice of the 99.9th percentile was determined empirically to minimise false positives whilst maintaining high detection rates for genuine anomalies.

5) *Anomaly Labelling Mechanism*: The trained autoencoder generates binary labels for all network traffic samples:

$$\text{Label}(x_i) = \begin{cases} \text{LEGIT} & \text{if } e_i \leq \tau \\ \text{ANOMALOUS} & \text{if } e_i > \tau \end{cases} \quad (20)$$

Samples with reconstruction errors exceeding the threshold indicate deviations from learned normal patterns, suggesting potential attacks or anomalous behaviour. This labelling mechanism is fundamentally different from traditional reconstruction error approaches that rely on aggregate metrics; our system evaluates per-sample errors independently, enabling detection of subtle deviations across individual features.

6) *Feature-Specific Anomaly Sensitivity*: Unlike methods that use a single aggregated reconstruction error, our approach captures feature-specific deviations. The reconstruction error for feature j of sample i is:

$$e_{ij} = (x_{ij} - \hat{x}_{ij})^2 \quad (21)$$

This granularity enables the system to identify anomalies characterised by deviations in specific features (e.g., unusual port numbers, atypical sequence numbers) even when other features remain normal. The total reconstruction error $e_i = \frac{1}{d} \sum_{j=1}^d e_{ij}$ provides an aggregate anomaly score whilst preserving sensitivity to individual feature anomalies.

7) *Supervised Classification Stage*: The autoencoder-generated labels serve as training data for a supervised classification model deployed to ESP32 devices. This second-stage model architecture comprises:

- Input layer: 10 features (matching autoencoder input)
- Hidden layer 1: 64 neurons with ReLU activation
- Hidden layer 2: 32 neurons with ReLU activation
- Hidden layer 3: 16 neurons with ReLU activation
- Hidden layer 4: 8 neurons with ReLU activation
- Output layer: 2 neurons with softmax activation (LEGIT-/ANOMALOUS)

The classification model is trained using sparse categorical cross-entropy loss:

$$\mathcal{L}_{\text{classification}} = - \sum_{i=1}^N \sum_{c=1}^2 y_{ic} \log(\hat{y}_{ic}) \quad (22)$$

where y_{ic} is the true label and $\hat{y}_{ic}$ is the predicted probability for class c . Training employs the same optimisation strategy as the autoencoder (Adam optimiser, early stopping, learning rate scheduling) with a maximum of 200 epochs and a batch size of 200.

8) *Advantages Over Traditional Methods*: Our autoencoder-based labelling mechanism offers several advantages:

- 1) **Unsupervised learning**: No manual labelling required; the system learns from raw network traffic
- 2) **Adaptability**: The threshold automatically adjusts to the statistical distribution of normal traffic
- 3) **Feature-level sensitivity**: Individual feature deviations are captured, improving detection of subtle attacks
- 4) **Computational efficiency**: Once trained, the labelling process requires only forward propagation through the autoencoder

5) **Transfer learning**: The labelled data enables training of lightweight classification models suitable for resource-constrained devices

This two-stage approach separates the computationally intensive unsupervised learning (performed centrally) from the lightweight classification (performed at the edge), enabling sophisticated anomaly detection on microcontroller platforms. Given the complexity of intrusion detection in IoT environments and the constraints of microcontroller-based devices, this work focuses on attack types that produce clear and measurable deviations in packet-level features. This design choice aligns with the capabilities of a lightweight, real-time classifier operating without payload inspection or cryptographic context.

J. Attack Detection

Our model is capable of detecting multiple categories of network-based attacks through the analysis of traffic patterns, including volumetric anomalies, temporal characteristics, address distributions, and TCP/IP header field behaviours. The primary attack classes detectable by the proposed system include:

a) *DoS and DDoS Attacks*: The system identifies DoS and DDoS attacks by detecting abnormal traffic volume surges targeting specific network destinations. TCP SYN Flood attacks are characterised by excessive incoming packets directed to a single destination IP address, exhibiting incomplete TCP handshake patterns. Detection is achieved by identifying high packet counts with sequence numbers but minimal or zero acknowledgement numbers, indicating half-open connection states. UDP Flood attacks are identified through large volumes of UDP protocol traffic directed at a single destination IP, typically targeting random high-numbered destination ports.

b) *Reconnaissance and Network Scanning Activities*: The model detects reconnaissance activities by analysing distinctive connection patterns. Port scanning (vertical scanning) is identified when a single source IP rapidly attempts connections to numerous destination ports on a single target IP, revealing service enumeration attempts. Host scanning (horizontal scanning) is detected when a single source IP probes identical port sets across multiple destination IPs, indicating network-wide reconnaissance activities.

c) *Evasion and Covert Channel Attacks*: The system identifies evasion techniques by detecting anomalous patterns in protocol usage. Non-standard protocol and port usage is flagged when low-volume traffic utilises uncommon protocol numbers, or when high-volume services operate on non-standard ports, potentially indicating command-and-control communications or data exfiltration attempts designed to circumvent conventional security controls.

Attacks such as man-in-the-middle and replay attacks often require cryptographic context, payload inspection, or timing-correlated session analysis. These capabilities exceed the scope of the current microcontroller-based IDS and are therefore left for future work.

K. Automatic Updates and Restarts

The ESP32 boards are configured for automatic updates, ensuring they remain secure and up to date. This functionality is achieved through:

Update Availability Check: The ESP32s periodically check for new security models and firmware updates from the designated servers. **Automatic Download and Installation:** Upon detecting available updates, the ESP32s automatically download and install them. **Self-Restart:** The ESP32s might automatically restart after installation to ensure the updates take effect.

L. Combined Network Awareness (Top Server)

The diagram depicts an additional server at the top level that combines the results from both segmented networks. This centralised server offers:

Comprehensive Threat Awareness: By combining the analysis from both networks, the system gains a broader understanding of potential threats across the entire testbed environment. **Improved Threat Response:** This comprehensive view allows for a more coordinated and effective response to security threats that might span across different network segments.

M. Overall Benefits

This testbed setup offers several advantages:

Proactive Security: The anomaly detection system identifies potential threats, enabling a proactive approach to security that prevents issues before they cause harm.

Optimised Performance: Batch traffic transmission and network segmentation contribute to more efficient network utilisation and improved device performance.

Automated Security Maintenance: Automated updates ensure that ESP32 devices consistently run the latest security patches and firmware, thereby minimising vulnerabilities.

N. Collaboration on the Edge

This system implements a decentralised classification mechanism in which multiple ESP32 devices collaborate to perform inference tasks using a neural network model. The approach is based on each device executing a segment of the overall inference process and exchanging the resulting outputs via UDP to build a consensus.

1) **Collaborative Inference:** Collaborative inference is implemented through a dedicated function that simulates the computations performed by different neural network layers. Each device processes a portion of the input data and generates final prediction outputs, which are then shared with peers. These final prediction outputs are later combined to produce the final classification output. This collaborative process leverages the outputs from multiple devices to compare predictions and enhance overall trust and accuracy.

2) **Packet Capture for Monitoring Network Traffic:** The system incorporates a packet capture mechanism operating in promiscuous mode, which allows devices to monitor and process network traffic. By analysing the TCP/IP headers of captured packets, a device can extract key information such as source and destination IP addresses, sequence numbers, and other relevant data. This information is then used to simulate network activity and populate the input features for the neural network. The system utilises the FreeRTOS task scheduler to enable concurrent classification and communication. A dedicated classification task, responsible for executing collaborative inference and managing communications, runs continuously on one core of the ESP32. The real-time operating system ensures that this task executes periodically, resetting the watchdog timer to prevent system crashes.

3) **System Architecture:** The system comprises three primary components:

- **Neural Network Classification:** ESP32 devices perform inference on input data using a TensorFlow Lite model.
- **UDP Communication:** Devices broadcast their inference results and receive predictions from other nodes to maintain agreement and trust.
- **Trust Evaluation:** Predictions are compared between devices; discrepancies are analysed to assess whether a device is trustworthy or potentially compromised.

4) **Neural Network Model:** The embedded neural network model runs on the ESP32 using the TensorFlow Lite framework. The model is configured with ten input features (N_INPUTS) and a single output (N_OUTPUTS), and it utilises a memory-efficient arena (TENSOR_ARENA_SIZE). The classification process is executed in two stages:

- 1) **Collaborative Inference:** Input data is processed in segments across different devices, each handling a prediction comparison.
- 2) **Final Prediction:** final prediction outputs are combined and passed to the network's prediction function to generate the final classification.

This staged approach enables the system to effectively manage resource constraints by breaking the workload into smaller, distributable tasks that can be handled across multiple devices.

5) **Collaborative Inference:** Collaborative inference is implemented via a dedicated function that simulates the computations performed by different neural network layers. Depending on the processing stage, a device handles a segment of the input data to generate final prediction outputs. These results are later combined to produce the final classification output. This collaborative machine-learning framework ensures that no single node is unduly burdened while maintaining the security and privacy of all involved parties.

6) **Communication via UDP:** Each ESP32 device broadcasts its collaborative inference results over the network using UDP, chosen for its low overhead and real-time transmission capabilities—vital for maintaining synchronisation between devices. Results are sent to all nodes via the broadcast IP address (255.255.255.255), facilitating local network communication. Upon receiving a packet, a device processes the incoming data and compares the result with its prediction. This

comparison, which involves calculating the absolute difference between the two predictions, determines trustworthiness. If the difference lies within the defined `TRUST_THRESHOLD`, the device is deemed trustworthy; otherwise, it is flagged as potentially compromised. Let $p_i(t)$ denote the local predicted anomaly probability (or score) at device d_i at time t , and $p_j(t)$ the score received from peer d_j . We define the deviation $\Delta_{ij}(t) = |p_i(t) - p_j(t)|$. A peer is considered consistent when $\Delta_{ij}(t) \leq \theta$, where θ is the `TRUST_THRESHOLD`. In our implementation, θ is selected empirically to balance tolerance to minor numerical variation against sensitivity to compromised or faulty outputs.

7) *Trust Evaluation*: The trust evaluation mechanism ensures data integrity within a decentralised network. Each device receives collaborative inference results from its peers and compares them to its predictions. The device is flagged as potentially compromised if the differences exceed the predefined `TRUST_THRESHOLD`. This process identifies possible faults or security breaches and assesses the system's overall resilience. Such a decentralised classification framework is particularly advantageous in IoT applications, where devices must collaborate to make decisions while distributing computational tasks and maintaining consistent, trustworthy outcomes, even in the event of individual node failures or compromises.

8) *Overview of Process*: As illustrated in Figure 3, the system operates through several stages. Each ESP32 device begins by capturing network data via packet capture. The device then processes this data locally and executes its portion of the collaborative inference using the embedded neural network model. Once an inference result is available, the device uses UDP to broadcast its predictions to other network nodes. Each device subsequently compares its predictions with those received from its peers to assess the trustworthiness of its own predictions. If a prediction significantly deviates from the others, beyond a predefined threshold, the peer device is flagged as potentially compromised. All ESP32 devices join a broadcast group, which enables them to discover one another even without prior knowledge of their IP addresses.

Devices regularly send their collected network data in chunks to an aggregation server. This server aggregates the data from all devices and uses it to train an updated machine-learning model. The newly trained model is then embedded in a firmware update made available by the server. Each ESP32 device periodically checks for firmware updates, downloads and installs any available updates, and reboots to resume operation with the latest model. This continuous cycle ensures the devices operate securely and remain synchronised with the most current model versions.

V. RESULTS

This section presents the performance outcomes from our experimental setup. We discuss several key metrics, including the evolution of the F-Score over time, inference time per device, network traffic patterns, and the relationship between training and validation performance. We aim to assess the system's reliability and practical feasibility under ideal and

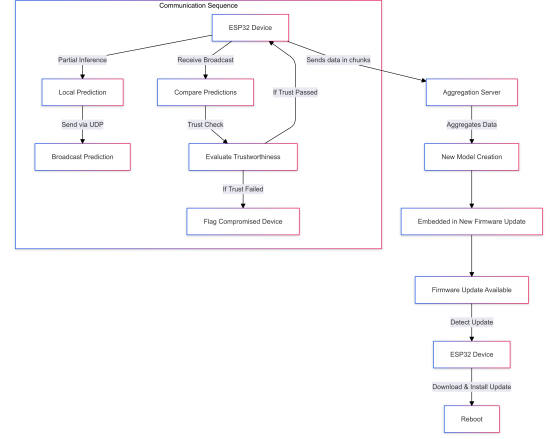


Fig. 3: Process flow of the testbed

challenging conditions, thereby identifying areas for further optimisation. The experiments presented in this section were conducted under controlled and relatively stable network conditions, as the primary goal of this work is to demonstrate a proof of concept for collaborative inference on resource-constrained devices.

A. F Score Over 24 Hours

Referring to Figure 4, the training conducted on an hourly basis resulted in a gradual improvement in the F Score, which eventually stabilised at 98.5 per cent. The steady increase in the F Score indicates that the system can effectively learn from the data. The minor fluctuations observed are mainly due to variability in the data captured by different devices. Although reflashing devices every hour is impractical for production deployments, we intentionally use hourly updates here as a controlled *stress-test* to observe convergence, stability, and the behaviour of asynchronous updates under frequent change. In practical deployments, updates would be scheduled less frequently and during low-activity windows.

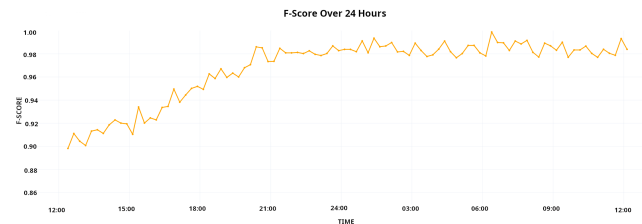


Fig. 4: F Score over 24 hours

B. Inference Time per Device

Figure 5 demonstrates that the inference time across devices remains generally low, which is an encouraging sign for real-time operation. On occasion, there are slight peaks in the recorded times. These peaks occur when a device ceases responding to inference requests due to the isolation mechanism implemented. When this occurs, the sending device experiences a brief delay upon detecting unresponsiveness,

after which it redirects its communication to another device. This observation confirms that our collaborative inference method is robust; however, it also suggests that the inference times may vary under different network conditions, such as those with slower or longer-range connections. Future tests under varying conditions will further validate these findings.

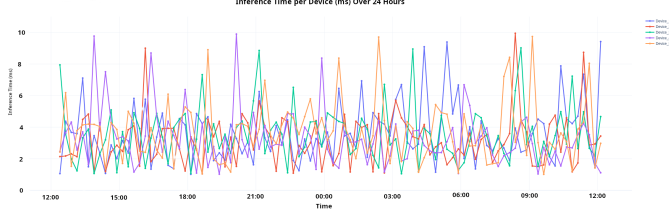


Fig. 5: Inference Time over Time

C. Traffic per Device

Figure 6 reveals a consistent pattern in network traffic across the devices. When a device fails to respond to a request, other devices attempt to verify this by retransmitting the data several times. This repeated transmission ensures that the lack of response is genuine and not a temporary glitch. In addition to these retransmissions, there are noticeable spikes in traffic, which can be attributed to periods during which devices transmit increased telemetry data. Such spikes are typical when a burst of information is generated, for example, during an event or when multiple devices synchronise to report their status. This pattern highlights the importance of effective traffic management, especially in deployments with limited network resources.

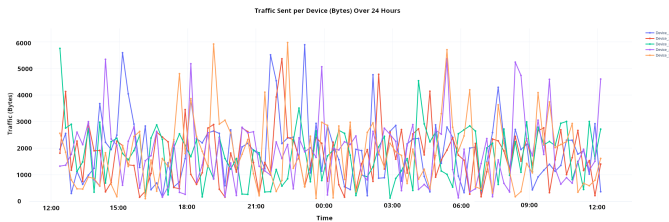


Fig. 6: Traffic per Device over Time

D. Observed Failure Modes

The following failure modes are relevant to real deployments:

(1) **Asynchronous update mismatches.** During frequent update cycles, devices may temporarily run different model versions. The model fingerprinting step detects mismatches and excludes affected nodes from collaboration until consistency is restored.

(2) **Packet loss and delayed peer responses.** UDP exchange occasionally experiences dropped packets or delayed delivery, resulting in transient disagreement. Devices mitigate this by periodically retransmitting and repeatedly inferring, which reduces the impact of single-message loss.

(3) **Unresponsive nodes.** When a device becomes unresponsive, peers retransmit for verification and then exclude the device once non-responsiveness persists, which appears in the inference-time peaks in Figure 5.

These observations motivate future evaluation under controlled packet-loss and congestion regimes to quantify degradation thresholds and validate robustness at scale.

E. Training versus Validation

Figure 7 illustrates the relationship between training and validation performance. The training accuracy (red line) consistently exceeds the validation accuracy (blue line), with this divergence increasing as training progresses. This indicates slight overfitting, where the model learns training data, including noise, reducing its ability to generalise to unseen data.

While effective mitigation of overfitting is crucial, resource constraints in IoT devices necessitate careful consideration. Complex mitigation techniques can impose significant computational overhead, straining limited resources, impacting device functionality, and increasing energy consumption.

Several strategies can be employed to address overfitting while respecting these constraints. Regularisation techniques, such as L1 and L2 regularisation [23], penalise model parameters to discourage overly complex relationships. Dropout layers [24] randomly deactivate neurons during training, promoting robust feature learning. Data augmentation increases training data diversity. Adjusting model complexity (e.g., reducing the number of parameters) and employing early stopping can also be beneficial.

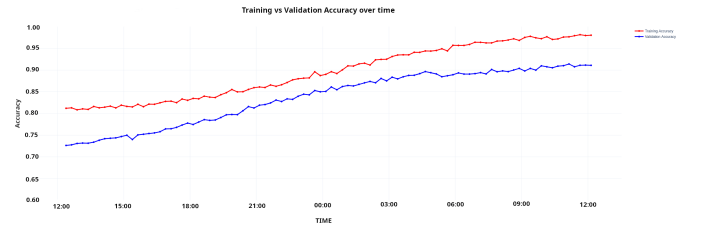


Fig. 7: Training versus Validation Performance

In conclusion, the experimental results presented in this section demonstrate that the system performs robustly under controlled conditions. The gradual improvement and high-stabilised value of the F Score, the low inference times, and the manageable network traffic collectively suggest that the system is well-equipped to handle typical operational challenges. Nonetheless, the variability observed across devices and the slight overfitting during training indicate that further refinements are needed. These include better scheduling of updates and mitigating overfitting to ensure consistent performance in real-world scenarios. Overall, these findings provide a solid foundation for the continued development and optimisation of our collaborative intrusion detection system.

F. Comparison with Existing Approaches

In Table III, we have compared our results with similar works. The 98.5% F-score achieved by our proposed model

demonstrates highly effective performance for an IDS operating within resource-constrained environments. When reviewing the comparison figures, the analysis must consider the operational context of each approach. Although the XGBoost-based IDS [25] records a slightly higher F-score of 99.5%, that model is explicitly designed for and deployed at the Cloud Level. This implies access to substantial computational and memory resources, rendering it inapplicable for direct deployment on low-power, Internet of Things (IoT) edge devices, which have strict hardware limitations.

The actual efficacy of the collaborative autoencoder is, therefore, best assessed against other models designed for similar hardware limitations. In comparison with the Anomaly-Based SNN [26], which was also implemented on comparable resource-constrained hardware (such as the ESP32), the SNN achieved an F-score of only 94.04%. Our 98.5% score from the collaborative approach represents a significant performance gain over this relevant edge-based baseline. This result confirms that the multi-layered, collaborative architecture successfully addresses the critical trade-off in IoT security: achieving a high detection rate and better overall accuracy without compromising the required resource efficiency. Finally, the lower CatBoost figure [27] of 85.08% is expected, as that model was evaluated on the notably more complex multi-class classification task.

TABLE III: F-Score Comparison of IDS Performance in Resource-Constrained Environments

IDS Model/Approach	F1-Score	Deployment Context
This Paper	98.5%	Edge/Binary
XGBoost-based IDS (Cloud Level) [25]	99.5%	Cloud/Binary (IDSH)
Anomaly-Based SNN (ESP32) [26]	94.04%	Edge/Binary (NSL-KDD)
Optimized CatBoost/SNN (Multi-class) [27]	85.08%	Edge/Multi-class

VI. LIMITATIONS OF APPROACH

The real-world deployment of resource-constrained IoT devices introduces several challenges that can significantly impact the performance and reliability of our collaborative intrusion detection system. We acknowledge that the model's performance would differ if it were run in a real-world environment, and that stronger hashes would be required in higher-risk or adversarial environments.

First, environmental unpredictability presents a substantial hurdle. IoT deployments frequently encounter unstable network connectivity, varying operational conditions (e.g., temperature fluctuations, power outages), and a diverse range of attack vectors. This inherent uncertainty demands that the system be highly adaptable and robust to avoid generating excessive false alarms, which can render the system unusable in practice. Furthermore, the dynamic nature of IoT environments can lead to concept drift, where the statistical properties of the data change over time, requiring continuous model adaptation [28]. While our evaluation demonstrates feasibility under controlled conditions, systematic testing in adverse,

unstable network environments remains part of ongoing and future work.

A significant challenge lies in scaling the system across a heterogeneous network. IoT ecosystems frequently comprise devices with vastly different hardware capabilities (e.g., processing power, memory), support various communication protocols (e.g., Wi-Fi, Zigbee, LoRa), and operate under diverse environmental constraints. These disparities necessitate careful planning for compatibility, seamless integration, and uniform performance across the network, a complex undertaking.

Timing and synchronisation issues further complicate the deployment of distributed IoT systems. Distributed devices, such as the ESP32 boards used in our testbed, can experience communication delays, especially when using protocols like UDP, which lack inherent reliability mechanisms. These delays can lead to out-of-sync operations and the exchange of outdated or incomplete data, compromising the accuracy of collaborative decision-making. While energy efficient, batch data transmission can introduce additional latency; if one device is delayed, the overall collaborative analysis may be compromised. Clock drift among devices, a common phenomenon in distributed systems, may exacerbate these timing inconsistencies, ultimately affecting data integrity and the reliability of trust evaluations [29].

Moreover, asynchronous updates of software or models present a security vulnerability. If segments of the network are running outdated detection logic whilst others are updated, this creates inconsistencies that attackers can exploit. Attackers might target vulnerable, unpatched devices to compromise the entire system [30]. This highlights the need for secure and atomic update mechanisms.

In addition to these challenges, collaborative intrusion detection systems are susceptible to specific security threats. Data falsification attacks, where malicious nodes inject false data into the collaborative process, can significantly degrade detection accuracy [31].

The system must incorporate several advanced mechanisms to mitigate the issues mentioned above. Reliable time synchronisation protocols are essential to minimise timing inconsistencies. Robust error correction and retransmission mechanisms for UDP communications can improve data delivery reliability. Coordinated update schedules and atomic update procedures are necessary to maintain system-wide consistency. Redundancy measures and fault-tolerance mechanisms can enhance system resilience. Furthermore, robust authentication and data validation techniques are crucial to defend against data falsification attacks.

Whilst our system employs UDP for lightweight multicast communication, we acknowledge that packet loss in unreliable network conditions warrants further investigation. Our collaborative verification mechanism incorporates inherent resilience through periodic retransmission: devices broadcast predictions every 3ms, and consensus is established over multiple inference cycles rather than single exchanges. This temporal redundancy means that occasional packet loss does not immediately compromise detection accuracy, as subsequent broadcasts enable devices to synchronise their collaborative decisions. However, systematic evaluation of system performance under

varying packet loss rates in controlled testbed environments would provide quantitative insights into the threshold at which detection degradation becomes significant. Future work could investigate adaptive retransmission strategies and alternative transport protocols for environments with high packet loss, as well as evaluate the trade-offs between UDP's low overhead and TCP's reliability guarantees in resource-constrained deployments.

VII. CONCLUSION

This paper presents a novel and practical collaborative intrusion detection system for resource-constrained IoT environments. By separating model training (centralised) and inference (distributed at the edge), our architecture enables efficient, low-latency local anomaly detection while minimising the computational burden on individual IoT devices. Integrating an autoencoder-based labelling system enhances the precision of threat identification, ensuring that only genuinely anomalous events trigger alerts, thereby reducing false positives. Our experimental results demonstrate that this approach strikes a balance between strong detection capabilities, low inference latency (critical for real-time security), and manageable network traffic overhead, even when deployed on resource-limited microcontroller platforms. This demonstrates the feasibility of deploying effective anomaly detection in resource-constrained IoT environments.

VIII. FUTURE WORK

Future research should prioritise several key directions. Firstly, enhancing adversarial resilience is crucial, requiring the development of robust mechanisms to detect and mitigate model poisoning attacks, as well as to detect anomalies in model updates, and to ensure model integrity at edge devices. Secondly, significant performance improvements can be achieved through optimised communication, necessitating an investigation into adaptive communication strategies that adjust protocol parameters based on network conditions (e.g., hybrid UDP/TCP, lightweight multicast protocols for IoT). Thirdly, broadening the applicability of our framework requires extending support for diverse IoT communication technologies (BLE, LoRaWAN, Zigbee), addressing technology-specific challenges in data aggregation, synchronisation, and security. Ultimately, long-term deployments and evaluations in real-world settings are essential to assess performance and robustness under realistic conditions, including device heterogeneity, dynamic networks, and evolving threats.

REFERENCES

- [1] Q. Li, L. Li, Z. Liu, W. Sun, W. Li, J. Li, and W. Zhao, "Cloud-edge collaboration for industrial internet of things: Scalable neurocomputing and rolling-horizon optimization," *IEEE Internet of Things Journal*, vol. 12, no. 12, pp. 19929–19943, 2025.
- [2] T. Luo and S. G. Nagarajan, "Distributed anomaly detection using autoencoder neural networks in wsn for iot," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.
- [3] H. Ochiai, R. Nishihata, E. Tomiyama, Y. Sun, and H. Esaki, "Detection of global anomalies on distributed iot edges with device-to-device communication," in *Proceedings of the Twenty-Fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, ser. MobiHoc '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 388–393. [Online]. Available: <https://doi.org/10.1145/3565287.3616528>

- [4] J. Ghadermazi, S. Hore, A. Shah, and N. D. Bastian, "Gtae-ids: Graph transformer-based autoencoder framework for real-time network intrusion detection," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 4026–4041, 2025.
- [5] M. Mainuddin, Z. Duan, and Y. Dong, "Detecting compromised iot devices using autoencoders with sequential hypothesis testing," in *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 2023, pp. 1344–1351.
- [6] G. A. Morales, J. Xu, D. Zhu, and R. Slavin, "Lightweight collaborative inferencing for real-time intrusion detection in iot networks," in *2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, MetaVerse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta)*, 2022, pp. 392–400.
- [7] Z. Tang, H. Hu, and C. Xu, "A federated learning method for network intrusion detection," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 10, p. e6812, 2022.
- [8] V. Muthukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X20329848>
- [9] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. Vincent Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.
- [10] N. F. Syed, M. Ge, and Z. Baig, "Fog-cloud based intrusion detection system using recurrent neural networks and feature selection for iot networks," *Computer Networks*, vol. 225, p. 109662, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S138912862300107X>
- [11] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X17308488>
- [12] S. T. Bakhsh, S. Alghamdi, R. A. Alsemmeiri, and S. R. Hassan, "An adaptive intrusion detection and prevention system for internet of things," *International Journal of Distributed Sensor Networks*, vol. 15, no. 11, p. 1550147719888109, 2019. [Online]. Available: <https://doi.org/10.1177/1550147719888109>
- [13] E. Anthi, L. Williams, and P. Burnap, "Pulse: An adaptive intrusion detection for the internet of things," in *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, 2018, pp. 1–4.
- [14] L. Wallgren, S. Raza, and T. Voigt, "Routing attacks and countermeasures in the rpl-based internet of things," *International Journal of Distributed Sensor Networks*, vol. 9, no. 8, p. 794326, 2013. [Online]. Available: <https://doi.org/10.1155/2013/794326>
- [15] T. Li, A. Rezaeiapanah, and E. M. Tag El Din, "An ensemble agglomerative hierarchical clustering algorithm based on clusters clustering technique and the novel similarity measurement," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, Part B, pp. 3828–3842, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1319157822001380>
- [16] I. Idrissi, M. Azizi, and O. Moussaoui, "A lightweight optimized deep learning-based host- intrusion detection system deployed on the edge for iot," *IJCDs Journal*, vol. 11, pp. 209–216, 01 2022.
- [17] M. Alazab, S. P. RM, M. Parimala, P. K. R. Maddikunta, T. R. Gadekallu, and Q.-V. Pham, "Federated learning for cybersecurity: Concepts, challenges, and future directions," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3501–3509, 2021.
- [18] M. Moshawrab, M. Adda, A. Bouzouane, H. Ibrahim, and A. Raad, "A maneuver in the trade-off space of federated learning aggregation frameworks secured with polymorphic encryption: Polyflam and polyflap frameworks," *Electronics*, vol. 13, no. 18, 2024. [Online]. Available: <https://www.mdpi.com/2079-9292/13/18/3716>
- [19] Y. Yao, Z. Xiao, B. Wang, B. Viswanath, H. Zheng, and B. Y. Zhao, "Complexity vs. performance: empirical analysis of machine learning as a service," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 384–397. [Online]. Available: <https://doi.org/10.1145/3131365.3131372>
- [20] A. Gerodimos, L. Maglaras, M. A. Ferrag, N. Ayres, and I. Kantzavelou, "Iot: Communication protocols and security threats," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 1–13, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2667345222000293>
- [21] A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of

- things,” in *2017 Internet Technologies and Applications (ITA)*, 2017, pp. 143–148.
- [22] D. Kataria, A. Yadav, and B. Singh, “Wireless communication systems: Bluetooth, infrared, zig bee, nfc and wi-fi outline,” *IJIRT*, vol. 1, no. 6, 2014.
 - [23] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 58, no. 1, pp. 267–288, 1996.
 - [24] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
 - [25] A. Javed, A. Ehtsham, M. Jawad, M. N. Awais, A.-u.-H. Qureshi, and H. Larijani, “Implementation of lightweight machine learning-based intrusion detection system on iot devices of smart homes,” *Future Internet*, vol. 16, no. 6, 2024. [Online]. Available: <https://www.mdpi.com/1999-5903/16/6/200>
 - [26] B. Boi, F. Cirillo, M. De Santis, and C. Esposito, “Anomaly-based intrusion detection system using esp32-wroom-da,” in *Advanced Information Networking and Applications*, L. Barolli, Ed. Cham: Springer Nature Switzerland, 2025, pp. 417–429.
 - [27] B. M. Kouassi, A. B. Ballo, K. J. Ayikpa, D. Mamadou, and Y. Diabagate, “Optimized intrusion detection in the iot through statistical selection and classification with catboost and snn,” *Technologies*, vol. 13, no. 10, 2025. [Online]. Available: <https://www.mdpi.com/2227-7080/13/10/441>
 - [28] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under concept drift: A review,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/8496795>
 - [29] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, p. 147–163, Dec. 2003. [Online]. Available: <https://doi.org/10.1145/844128.844143>
 - [30] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, “Security, privacy and trust in internet of things: The road ahead,” *Computer Networks*, vol. 76, pp. 146–164, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128614003971>
 - [31] M. A. Rahman and H. Mohsenian-Rad, “False data injection attacks with incomplete information against smart power grids,” in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 3153–3158. [Online]. Available: <https://ieeexplore.ieee.org/document/6503599>