SN Computer Science (2026) 7:186
https://doi.org/10.1007/s42979-026-04750-x

ORIGINAL RESEARCH

A Cloud-Driven Support Layer for Enhancing Distributed IDS in IoT Networks

Gustavo Funchal[1,2] · Tiago Pedrosa[1,2] · Fernando De la Prieta[3] · Paulo Leitão[1,2]

Received: 4 August 2025 / Accepted: 10 January 2026

Abstract

The exponential growth of connected devices, including sensors, mobile equipment, and various Internet of Things (IoT) nodes, has significantly increased the volume of data generated at the edge. Traditionally, data analysis tasks are offloaded to centralized cloud servers, resulting in increased latency, bandwidth bottlenecks and privacy concerns. While edge computing addresses these limitations by enabling local processing, it also faces challenges related to limited computational capacity and isolated decision-making. In this context, Multi-Agent Systems provide a promising solution by enabling collaboration among edge nodes for distributed machine learning-based intrusion detection. This work extends previous research by introducing a hierarchical approach within the edge-cloud continuum, where agents deployed in the cloud continuously monitor edge-level behaviour and employ reinforcement learning techniques to suggest dynamic updates to decision parameters of edge agents. This feedback-driven mechanism allows agents to adapt their behaviour over time, improving detection accuracy and collaboration efficiency while keeping communication overhead under control. The proposed architecture balances decentralisation and adaptability, offering a scalable and privacy-preserving solution for intrusion detection in dynamic and resource-constrained IoT environments.

Keywords Internet of Things · Multi-Agent Systems · Reinforcement Learning · Intrusion Detection Systems

Introduction

The implementation of Industry 4.0 involves linking corporate assets to the Internet to collect extensive data, facilitating the creation of innovative services and apps that improve operational efficiency and effectiveness [1]. In this context, the integration of cyber-physical systems (CPS), combined with Internet of Things (IoT) technologies and Artificial Intelligence (AI) techniques, plays a crucial role. CPS refers to the management/control of systems that combine hardware and software in a collaborative networked structure, while IoT emphasizes the digitization of products and resources by deploying small sensors connected to the Internet throughout the system.

These internet-connected devices, called edge devices, provide an entry point into the main networks of companies or service providers, and are located close to the data source [2]. The edge computing paradigm comes from performing computing/processing close to the data source, without sending data to external service providers (e.g. cloud), making data processing more reliable and secure in the case of critical systems, or those containing sensitive data, such as

Tiago Pedrosa, Fernando De la Prieta and Paulo Leitão have contributed equally to this work.

✉ Gustavo Funchal
gustavofunchal@ipb.pt

Tiago Pedrosa
pedrosa@ipb.pt

Fernando De la Prieta
fer@usal.es

Paulo Leitão
pleitao@ipb.pt

1 Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Campus de Santa Apolónia, Bragança 5300-253, Portugal

2 Laboratório para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC), Instituto Politécnico de Bragança, Campus de Santa Apolónia, Bragança 5300-253, Portugal

3 BISITE Research Group, University of Salamanca, Edificio I+D+i, C/ Espejos s/n, 37007 Salamanca, Spain

Published online: 07 February 2026

hospitals, factories, and retail locations [3]. It also reduces latency and optimizes bandwidth. However, despite the growing feasibility and advantages of local data processing at the edge, cloud computing continues to play a key role within the Industry 4.0 ecosystem. The cloud provides virtually unlimited computational power, large-scale storage capabilities, and access to advanced services such as AI model training and global data analytics, which are often beyond the scope of resource-constrained edge devices.

Therefore, although reducing reliance on the cloud can improve responsiveness and enhance data sovereignty, completely eliminating cloud services is neither practical nor efficient for many industrial applications. This scenario has led to the emergence of a collaborative paradigm known as the edge–cloud continuum, where tasks and responsibilities are intelligently distributed along a spectrum that spans from edge devices to centralized cloud infrastructure. The edge–cloud continuum enables a flexible and context-aware computing model, allowing data processing to occur at the most appropriate level depending on factors such as latency requirements, privacy constraints, processing power availability, and communication costs. This integrated approach combines the strengths of both edge and cloud computing, fostering the development of robust, scalable, and intelligent Industry 4.0 systems.

As the number of connected devices continues to grow exponentially, the amount of data generated and the communication between systems also increases significantly. This intensification of digital traffic imposes new challenges related to cybersecurity, especially in industrial and mission-critical environments. In this scenario, leveraging both edge and cloud computing enables the design of multi-layered security strategies. The edge offers low-latency processing, improved data privacy, and optimized bandwidth usage, while the cloud provides scalable computational power for complex analytics and decision-making tasks.

One of the critical aspects of cybersecurity in IoT environments is the timely detection and mitigation of cyber threats. Intrusion Detection Systems (IDS) play a pivotal role in safeguarding industrial systems against unauthorized access, malicious activities, and potential disruptions [4]. IDS based on distributed agents at the edge have already been proposed in previous works [5, 6], mainly due to their potential for real-time response and system autonomy. However, the recent advances in Machine Learning (ML) open new opportunities to enhance these agents by embedding intelligent decision-making capabilities into them. With ML support, agents become more autonomous, context-aware, and capable of making accurate security decisions, especially when deployed across different levels of the edge–cloud continuum.

Although previous work [6] has shown excellent results, in which agents collaborate at the edge to increase performance in detecting attacks, there are still certain limitations, such as when aggregating the results of collaboration between different agents, the parameters used to calculate the final decision are static, meaning that there is no evolution throughout the process. Furthermore, in a dynamic environment, each agent may perform better in a different way when aggregating data and requesting help for a collaboration. In addition, with static parameters, there may be bottlenecks with too many requests for help from the agents, interfering with the quality of the network they are connected to due to the high flow of message exchanges.

Having this in mind, this work proposes the evolution of the architecture based on Multi-Agent Systems (MAS), where intelligent agents located closer to the devices/assets (e.g. in the edge) collaborate to detect anomalies and cyberattacks in IoT networks. The main contribution of this research lies in the introduction and implementation of a dynamic and hierarchical adaptation mechanism, achieved through the inclusion of a higher-level support layer that operates on the edge application layer. While previous work focused only on static collaboration at the edge, this new upper layer employs Reinforcement Learning (RL) techniques to analyse the aggregate behaviour of agents at the edge and continuously suggest dynamic updates to the decision parameters, used in decision-making process. This adaptive feedback-based system overcomes the limitation of static values, allowing the system as a whole to evolve and optimise detection accuracy and collaboration efficiency over time. This proposal for higher-level analysis had been mentioned in previous works, but it is in this present work that it was specified, implemented, and validated, demonstrating promising results in improving the capabilities of IDS in distributed industrial environments.

The remaining article is organized as follows: Sect. 2 reviews related work on distributed analysis for IDS, with a focus on edge computing, cloud environments, and the edge–cloud continuum in IoT environments. Section 3 introduces the proposed architecture, detailing its two-layered structure, System Support Layer and System Application Layer and, the interactions among distributed agents. Section 4 presents the reinforcement learning model employed in the System Support Layer agents, formulated as a Markov Decision Process. Section 5 describes the experimental setup and the case study used to validate the proposed approach. Section 6 discusses the results and the insights gained. Finally, Sect. 7 concludes the article and outlines directions for future work.

Related Work

The generalized and exponential adoption of smart IoT devices, especially in industrial environments, is accelerating the search for new techniques to make IoT applications secure, scalable and energy-efficient [7]. IoT devices typically come equipped with various sensors that gather environmental/operational data, serving as key components of data-driven intelligence systems [8]. As these devices' deployment expands, the generated data volume grows exponentially. In order to provide insights to end users, it is necessary to process this collected data and analyze it first. Moreover, internet traffic between devices in an IoT network must be monitored, commonly by Network Intrusion Detection Systems (NIDS), acting as a first line of defense in order to identify potential threats and protect the network from malicious attacks and intruders [9]. However, most IoT devices have limited computing resources, making this processing a major challenge.

A commonly adopted solution is cloud computing, where IoT data is sent to remote servers for processing, and the results are transmitted back to the devices. While effective, this approach can face significant challenges in terms of data transmission rates and network bandwidth, which can become critical bottlenecks as IoT ecosystems scale [10]. In addition, as IoT devices often handle personal and sensitive data, routing all information to cloud servers raises security and privacy concerns [8] and when these devices require a high service response time, it becomes a major challenge for cloud-based IoT applications [11].

In this context, edge computing is gaining attention, being a type of IT architecture in which data is processed at the edge of the network, or as close as possible to the data source, reducing costs and response times, increasing data privacy and security, and making it possible to make decisions in these network applications faster and with lower response latency [11]. Despite the numerous advantages of edge computing over cloud computing, it cannot fully replace cloud services [12, 13]. While shifting analytics to the edge network is designed to reduce service response times, certain services still depend on cloud infrastructure. Moreover, the edge computing layer encounters several challenges, including task offloading, performance optimization, energy efficiency, Quality of Service (QoS) support, and connection management [14, 15].

Some benefits of edge computing were highlighted by [11], such as cost savings, backhaul traffic reduction, improved QoS, enhanced network customization, and improved service response times and distribution capabilities, which justify the growth in the adoption of edge computing. However, as the complexity of IoT applications increases, especially in environments such as Industrial Internet of Things (IIoT), traditional centralized ML approaches become insufficient due to computing power constraints and the need for real-time analysis. This has led to the development and integration of distributed ML, where computational tasks are spread across multiple nodes, enabling large-scale data processing and model training closer to the data source.

When dealing with this type of distributed solution, the need to exploit the advantages of both edge computing and cloud computing becomes evident. In this context, the concept of edge-cloud continuum emerges, which represents the continuous and dynamic space between the data source (typically at the edge) and the processing centres (typically in the cloud). As pointed out by [16], this continuum can take different forms, ranging from direct connections to architectures made up of multiple layers and jumps, with a wide diversity of technologies involved. Regardless of physical proximity to the cloud or the computing capacity available at each layer, the central idea is that there is a range of intermediate possibilities that can be explored. Thus, intelligent utilisation of this continuum allows for more efficient and adaptable solutions, especially in scenarios with specific restrictions at different levels of the application. Figure 1 illustrates the various levels of the continuum, highlighting their typical characteristics and actions, from those closest to the operational environment to those closest to the centralised high-performance infrastructure.

Also, distributed ML offers the potential to overcome the computational bottlenecks that arise from handling large data sets and complex models at the edge. By distributing the training and inference processes across several edge nodes, it becomes possible to leverage the collective resources of the network, resulting in enhanced scalability, reduced latency, and improved adaptability to dynamic environments. Additionally, distributed ML techniques are particularly effective in scenarios where data privacy is crucial, as the data can remain localized at the edge nodes, minimizing the need for transmission to centralized servers. There are some strategies that can be used for distribution, such as data parallelism, model parallelism, ensemble learning/model combination, and model diversity. However, there are several issues that need to be analysed and addressed, namely minimal synchronization, communication overhead, device heterogeneity, security and privacy, among others [18, 19].

As highlighted in [20, 21], data parallelism divides a batch of data into several smaller batches, and these input data samples are distributed among several computing resources (nodes or devices). It can thus improve the performance of large batch workloads. Each device performs local processing with its own data and the complete model, a copy of which is stored locally. Because the full model is present on every device, this structure works well with a

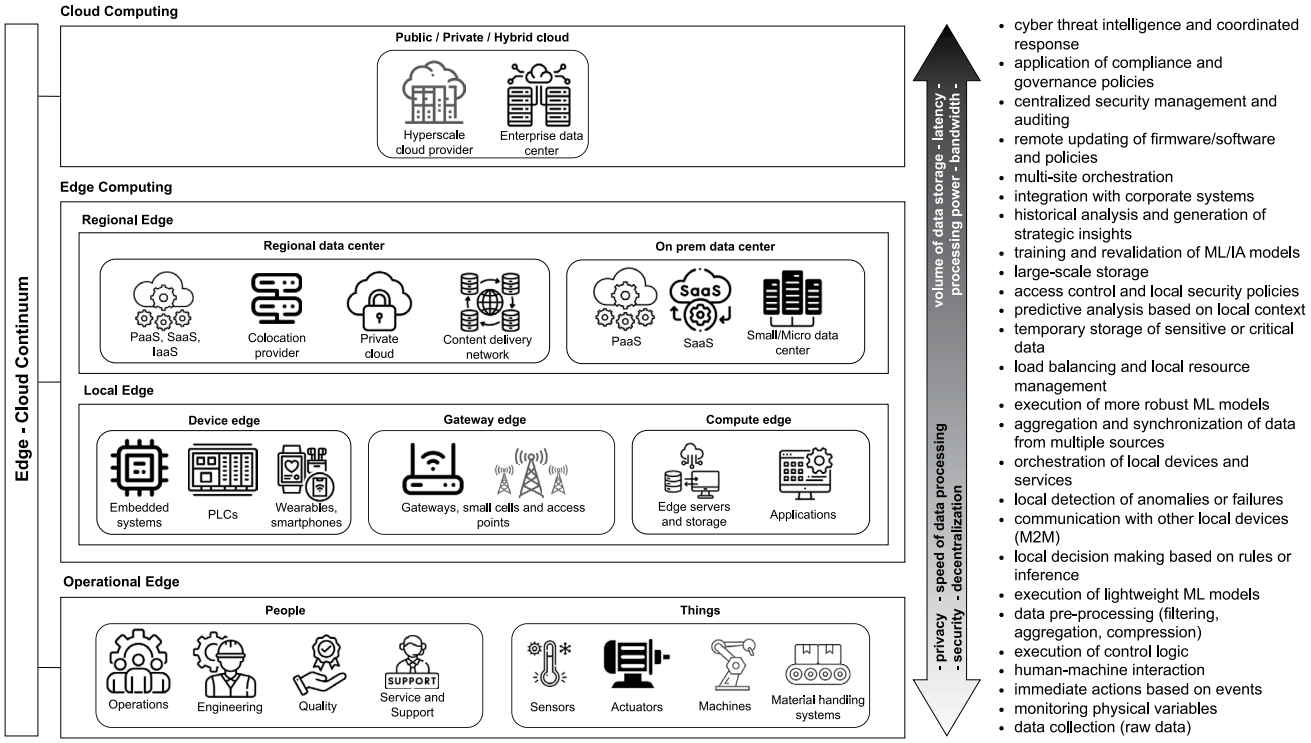


Fig. 1 Edge-cloud continuum overview (adapted from [17])

wide range of model topologies and scales effectively when the model has few parameters. However, the parameter synchronization becomes a bottleneck when the model has many parameters [22].

On the other hand, [20] notes that in the model parallelism a part of the model that each device uses for local processing is stored on it. Every device has a duplicate copy of the data. Due to its limited memory footprint and consequently low memory requirements on each device, this strategy works well when the model is too big to fit on a single device. Effective model splitting can be difficult, though, as a poorly done split can cause synchronization delays and communication overload, which can cause downtime [20, 23].

In addition to these strategies, there is ensemble learning [24], in which each node trains an independent model and then these models are combined to form a final model. This final model can be made up of different characteristics, which can enhance the results obtained by taking advantage of the strengths and reducing the weaknesses of each model. There are different possible techniques, such as bagging (or bootstrap aggregating), boosting and stacking that can be explored.

ML algorithms are used in IDS because of their capacity to recognize patterns, correlate events and adjust to emerging threats [25, 26]. The main benefits of using ML in IDS is the high anomaly detection accuracy, scalability to manage big data sets, and the potential to get better with time and new information. These technologies improve security by reducing false positives and providing real-time threat detection.

In this respect, many works have been developed in this area, such as the work proposed by [27], which highlights the use of ML algorithms, namely Random Forest, in which they propose an edge approach with methods for feature selection, dimensionality reduction and outlier removal in order to reduce computational costs and time, and improve performance in Industrial IoT intrusion detection, and also the work carried out by [28], that proposed a Temporal Convolutional Network (TCN) based intrusion detection method for IoT environments that is lightweight, effective and relies on cloud edge collaboration, based on a federated learning framework. To this end, they have also reduced the dimensionality of the high-dimensional resources of raw network traffic data to reduce computing and storage requirements while overcoming the problem of resource limitations of edge devices. [28] also carried out some experiments to validate that the collaboration-based approach at the cloud edge can share threat intelligence and has the potential to defend against unknown attacks in a collaborative way, showing that collaboration was extremely important and could help participants identify their own unknown attacks.

However, there are still other distributed approaches, such as those based on MAS [29], which act as containers/

repositories for AI algorithms [30]. According to [31], an agent can be defined as an autonomous entity, which represents a physical or logical part of the system, and which will be able to take actions to achieve the system's objectives and will also be able to interact with other agents in the system when it does not have the skills to achieve its objectives alone. Thus, the use of MAS allows to distribute intelligence and adaptation [32], in which decisions are made in a decentralized manner, as opposed to centralized structures that are unable to meet the requirements related to response time, data privacy and security, network bandwidth, among others. Although there is a certain complexity to coordinating the actions of various agents in a distributed environment, it becomes very advantageous to use it in dynamic systems, especially in IoT environments, where conditions change rapidly, there is a great heterogeneity of devices, which can use different algorithms due to their available computing resources.

Modern IoT applications, especially in industrial environments, have driven a trend towards the use of the edge-cloud continuum [33], in which intelligence is distributed across multiple processing levels, from edge devices to cloud servers. This layered architecture makes it possible to take advantage of the specific characteristics of each level, namely the reduced latency and local context of the nodes at the edge, combined with the computing power and global vision of the cloud. Recent work has emphasized the importance of adopting hybrid and hierarchical approaches, capable of making decisions in an adaptive and coordinated way between the different levels of the architecture, in order to balance efficiency, scalability, privacy and responsiveness.

In this context, MAS stand out as a promising approach to modelling autonomy and collaboration between different levels of the hierarchy, making it possible to build more flexible and resilient architectures. The decentralized nature of MAS makes them ideal for the edge-cloud continuum, since each agent can make decisions locally, collaborate with other agents, and still respect restrictions such as sensitive data privacy and device heterogeneity. In addition, agents located in the cloud with greater computing power can act as support agents, helping agents at the edge with suggestions or parameter adaptations, without having to directly access local data. This approach preserves privacy and introduces a layer of adaptive intelligence based on indirect observations (e.g. performance reports), paving the way for the use of Reinforcement Learning (RL) algorithms and other distributed optimization techniques.

Despite the fact that the aforementioned works have presented excellent solutions and results, the distribution of intelligence across multiple layers of the edge-cloud continuum remains underexplored, especially regarding agent-based approaches capable of enabling decentralized, privacy-preserving, and adaptive decision-making. In particular, MAS presents a promising strategy not only for collaboration within the edge network, but also for orchestrating intelligent support of the cloud layer. In this extended context, the proposed work investigates different strategies for distributing intrusion analysis and detection in the edge and cloud layers, taking advantage of MAS as a coordination framework. The main focus is to present the design and implementation of a module based on RL in the superior layer (e.g. cloud) that continuously evaluates reports from edge agents and suggests parameter adjustments to improve their performance, all without direct access to local data. This contribution complements previous results on edge-level strategies and advances the discussion towards a more holistic and hierarchical distributed intelligence.

Distributed Attack Detection in IoT

This work builds upon the previously presented foundational architecture in [5, 6], which was structured around a traditional edge–cloud paradigm. While the earlier approach successfully demonstrated how MAS could be used to distribute intelligence across the edge–cloud continuum, the current architecture introduces a more abstract and modular design based on two distinct functional layers: the *System Application Layer* and the *System Support Layer*. This updated structure aims to better capture the logical separation between operational processes and decision support services, while preserving the interactions and collaborative behaviours described in the original model.

The System Application Layer encompasses the edge components, where Local Agents (LAs) are responsible for data collection, initial analysis, and inter-agent collaboration under resource constraints. The System Support Layer assumes the role previously attributed to the cloud, hosting more powerful Global Agents (GAs) that provide strategic assistance and recommendations without directly accessing sensitive local data. Although the terminology and abstraction have evolved, the key communication patterns, between edge agents, edge agents and cloud agents, and also between cloud agents, remain consistent with the previous architecture. For further details regarding agents communication, collaboration protocols, and edge layer distribution strategies, is described in the previous work [6].

The redesigned architecture facilitates the integration of RL mechanisms in the System Support Layer, enabling adaptive system optimization through feedback driven decision making. The focus of this paper is on the new RL module embedded in the GAs, and its ability to influence decision making at the edge through parameter adjustment and tactical guidance.

System Architecture

The proposed system architecture is made up of two layers, the System Support Layer and the System Application Layer, as shown in Fig. 2. This architecture integrates multiple agents across both layers, with LAs that are linked to processes or systems in a manufacturing environment, to make it possible to digitize these assets and collect information to enable analyses that improve the efficiency and effectiveness of operational systems. LAs are part of the cyber layer of a cyber-physical component/system, representing the logical part of this component and, in a way, where it is possible to make it 'intelligent'. In this architecture, the LA has some associated modules, e.g. the generation of behavioural reports, the execution of local IDS, interaction and communication with other agents.

On the other hand, the GA, which are in the System Support Layer, are not directly linked to the operational part of the system, but are computational entities with high computing power that communicate with the LAs to provide

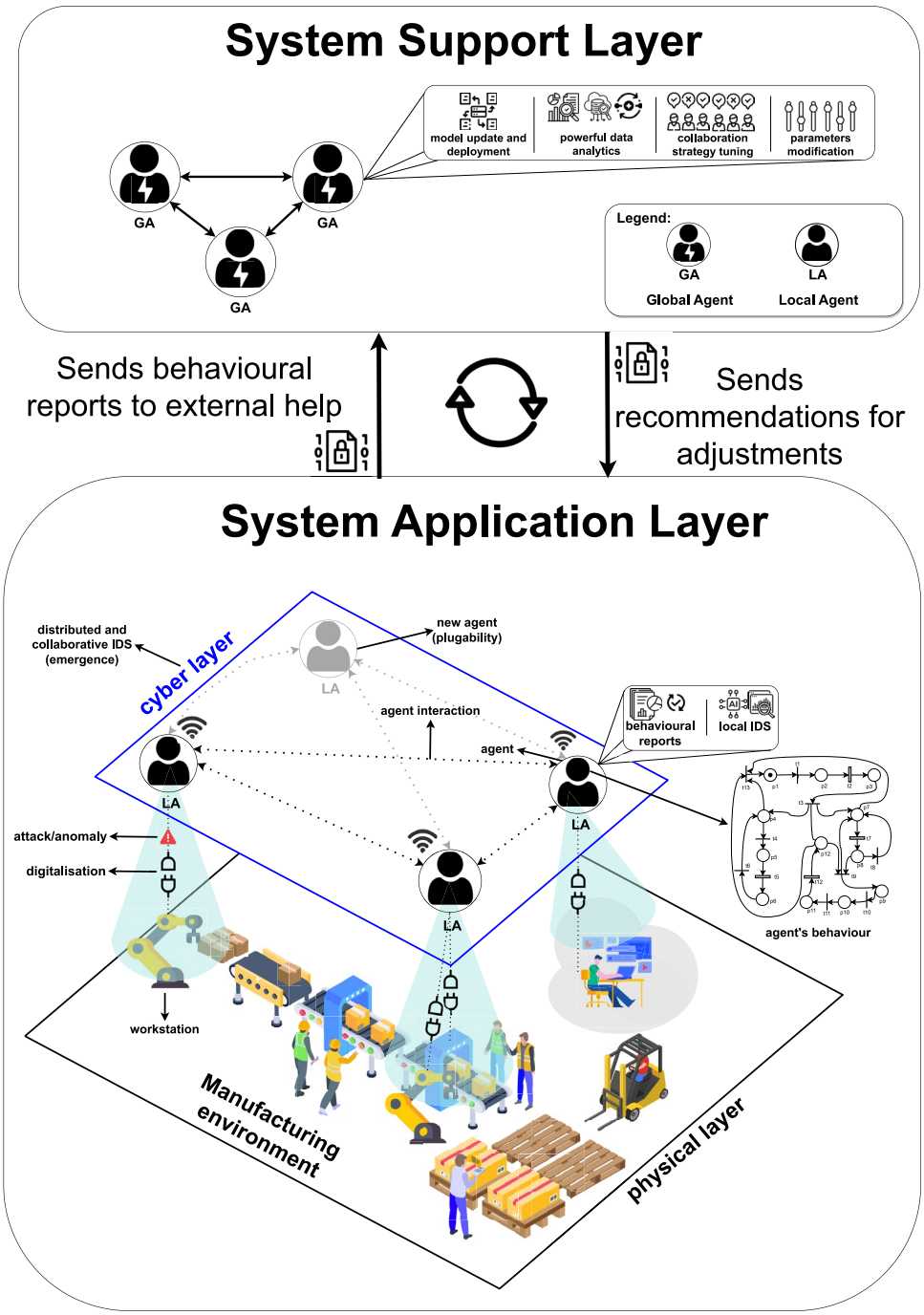


Fig. 2 System architecture composed of agents distributed along the two layers *System Support* and *System Application*

assistance or recommendations for improvement. GAs have powerful data analysis capabilities, update models or model parameters, improve collaboration strategies, among others.

System Application Layer

The System Application Layer comprises the layer where the operating systems of a given manufacturing environment are located. There is all the machinery (the physical part of the system), as well as the cyber part (the logical part of the system).

In general, LAs in the System Application Layer are directly linked to operational processes. These LAs are typically executed on edge computing devices (e.g., gateways or even directly on IoT devices used in operational processes) to ensure low-latency data processing. They continuously collect data from the sensors, carry out local data analysis using embedded ML models, and collaborate with each other to achieve better results (i.e. improving detection accuracy, especially by minimizing false negatives and avoiding incorrect classifications), but they have restrictions in terms of computing power. To maintain data privacy, operational data is always kept in this layer. The LAs generate behavioural reports to report on the performance of their analysis, the rate of collaboration that is required (measured in relation to the number of requests that are made to other agents in a given period), the resources used, to map whether it is balanced throughout the system and whether it is maintaining the security of the system, and this behavioural report is shared with the GA which has high capacity in analysis and will take certain actions to suggest recommendations for obtaining better results. Reports will summarize the behaviour of LA, in terms of collaboration requests, probabilities of predictions, etc. This interaction between LAs is described in the diagram shown in Fig. 3.

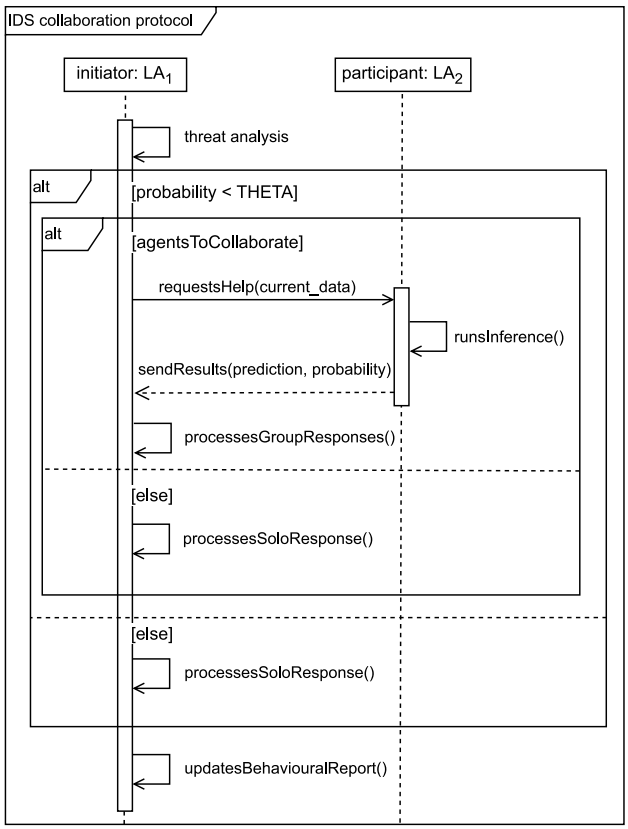


Fig. 3 Interaction protocol between local agents for collaboration in an IDS

It is important to note that when an LA receives support from other agents, it must aggregate the responses to make a final classification decision. This process is triggered when the LA's initial confidence is lower than a decision threshold θ, and collaboration is permitted (i.e. there are agents to collaborate). Otherwise, the agent applies its own prediction directly. These two situations are represented in the UML diagram in Fig. 3, specifically in the methods *processesSoloResponse* and *processesGroupResponses*.

In collaborative scenarios, the final decision is computed through a weighted aggregation of peer responses, as formalized by the following equation:

$$\hat{y} = \begin{cases} 1, & \text{if } \left(\frac{\sum_{i=1}^{n} (w_i \cdot \hat{y}_i \cdot \mathbb{I}(C_i \geq \theta))}{\sum_{i=1}^{n} w_i} \right) \geq \alpha \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where $\hat{y}_i$ is the binary prediction received from agent i, C_i is the confidence associated with that prediction, w_i is a weight assigned to agent i (e.g., based on performance metrics such as accuracy), $\mathbb{I}(C_i \geq \theta)$ is an indicator function that filters out responses with low confidence, α is the aggregation threshold used to determine whether the final weighted vote supports the class representing the attack.

If the weighted average of the trusted responses reaches or exceeds α, the result is classified as attack. Otherwise, it is classified as normal/benign. This fusion mechanism enables agents to dynamically adapt their collaboration strategy based on both peer confidence and system-wide tuning parameters. More detailed discussions on the collaborative protocol and parameter behaviour are available in the previous work [6].

System Support Layer

The System Support Layer represents a high-performance/powerful computing layer, which could be, e.g. a powerful computing platform or even a public, private or hybrid cloud server.

The GA presented in this layer has several skills, including suggesting/recommending certain actions, mainly

concerning tactics, strategies, assistance and replacement, namely:

- **Tatics**: is related to which tactic the LA will use to aggregate the data received when they request help between LA, and can be switched between voting (majority wins), averaging, weighted averaging, or even changing the parameters used to calculate the aggregation of data received.
- **Strategies**: are related to which models will be used by the LA, which can be specialized models for certain types of attack, or generalized models, with a vast knowledge of different attacks. Specialized models may have better detections for specific attacks, but may depend on more collaboration because they have a gap for other types of attacks, while general models may have more affected detections because they are general models, but with collaboration they can cover this gap.
- **Assistance**: is related to changing the parameters of the model, i.e. changing the hyperparameters of the model used by the LA.
- **Replacement**: is related to the exchange of the model used, i.e. it will be suggested that the model used be replaced by another that may perform better.

Although various types of actions that the GA can carry out in the system have been described previously, such as helping with model selection (strategies), adjusting the model's hyperparameters (assistance) or suggesting model replacement, the focus of this work is specifically on tactical decisions, particularly adjusting the parameters used to aggregate the responses of other LA when collaboration is triggered.

In this context, it will be explored how GA can dynamically suggest updates to the decision parameters employed by each LA when aggregating the results of peer responses. Specifically, the LA sets a threshold θ to determine whether its own prediction is sufficiently reliable. If not, it requests help from neighbouring LAs. Once the responses have been received, the final decision is made based on a weighted aggregation that considers both its own prediction and the predictions of its peers, modulated by a weight parameter α. Thus, θ and α directly affect the tactic used to combine information: θ controls when collaboration takes place, while α defines how much the LA trusts its own prediction compared to the assistance received.

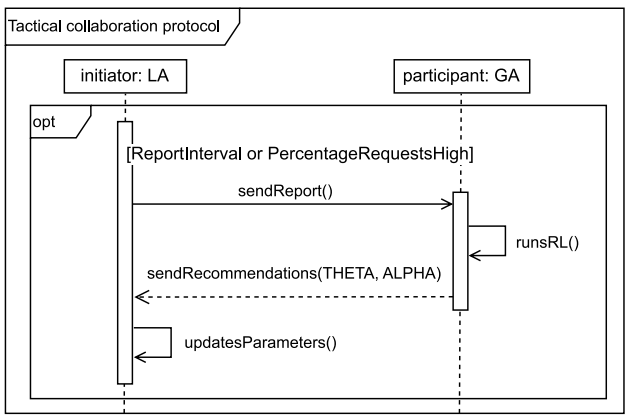


Fig. 4 Interaction protocol between local and global agents to adjust decision parameters

By intelligently adjusting these parameters through RL in the System Support Layer, the system is able to adapt over time to the performance of each agent, the conditions of the network and the dynamics of the attack. Therefore, although various types of GA interventions are possible, this work focuses on optimising the decision fusion process, which is a central element of the system. The impact of this actions will be the resolution of conflicts (high collaboration request rate, performance of attack detection, etc). The interaction between LA and GA is described in the diagram shown in Fig. 4.

Communication Triggers for Report Transmission

In this architecture, LAs continuously monitor their local performance and periodically generate structured reports containing relevant metrics. Each report is formatted as a JSON object and includes key operational indicators that allow the GA to assess both the individual agent's behaviour and the overall system effectiveness (see Table 1). Specifically, the report includes the current decision thresholds (θ and α), the local model's accuracy, the number of false positives and false negatives (both for the current interval and in total), the number of detected attacks, the total and interval-based number of processed packets, the number of collaboration requests, as well as system-level metrics such as memory usage, CPU usage, latency, timestamp, and a unique `agent_id` field for identification to associate the report with the correct agent in the distributed environment.

This structured reporting mechanism enables the GA to make targeted decisions and track trends over time. However, to avoid unnecessary communication with the GA, the system defines two communication triggers responsible for controlling when these reports should be transmitted. The first trigger ensures that regardless of the system's behaviour, a report is sent periodically in order to keep the system up to date, while the second trigger seeks to identify when an agent is requesting too much help, which could indicate inefficiency in the current model or trust issues, in addition to giving the agent sufficient autonomy to determine when updates are necessary. The richness of the report data, combined with well-defined triggers, supports a responsive and adaptive RL mechanism in the System Support Layer.

Table 1 Structure of the report sent by local agents

Field	Description
`agent_id`	Unique identifier of the local agent
`timestamp`	Time when the report was generated
`decision_threshold_theta`	Current θ threshold used for initial decision
`final_decision_threshold_alpha`	Current α threshold used for final decision
`model_accuracy`	Model accuracy based on recent predictions
`false_positives`	False positives during the current interval
`false_negatives`	False negatives during the current interval
`detected_attacks`	Number of detected attacks in the interval
`processed_packets`	Packets processed in the interval
`false_positives_total`	Total false positives accumulated
`false_negatives_total`	Total false negatives accumulated
`processed_packets_total`	Total processed packets accumulated
`collaboration_requests`	Number of collaboration requests during the interval
`cpu_usage`	Percentage of CPU usage
`memory_usage`	Percentage of memory usage
`latency_ms`	Average latency (in ms) during inference

Trigger 1: Sample-based Periodicity

Let P be the number of samples processed since the last report, and N be the configured periodicity. Trigger 1 is activated when the number of analyzed samples reaches the periodic threshold:

$$P \bmod N = 0 \tag{2}$$

This ensures that the agent reports periodically, even under stable operating conditions, allowing the global agent to monitor system behaviour and learning dynamics over time.

Trigger 2: Collaboration Rate Threshold

Trigger 2 is designed to activate in scenarios where the collaboration rate between agents increases significantly. Let C be the number of collaboration requests issued in the current window of analysis (since the last report), and let P again be the number of processed samples in that window. The collaboration rate is defined as:

$$T_c = \frac{C}{P} \cdot 100 \tag{3}$$

To avoid premature activations due to small values of P, it was defined a minimum number of samples $P_{\min}$ necessary for Trigger 2 to be valid. This minimum is calculated as a fraction of the periodicity N, using a sensitivity factor $\beta \in (0, 1)$:

$$P_{\min} = \beta \cdot N \tag{4}$$

The second trigger is activated if and only if the collaboration rate exceeds an acceptable threshold T_{allowed}, and the number of processed samples is sufficient:

$$T_c \geq T_{\text{allowed}} \quad \text{and} \quad P \geq P_{\min} \tag{5}$$

Final Triggering Condition

A report is sent to the global agent when either of the two triggers is satisfied:

$$\text{Send\_report} \iff \text{Trigger}_1 \vee \text{Trigger}_2 \tag{6}$$

This combined strategy provides a balance between periodic monitoring and responsiveness to dynamic collaboration patterns, ensuring timely adaptation of the decision parameters (θ, α) according to the behaviour of the distributed system. Figure 5 shows an example of how the two triggers work together. In this example, it was assumed that every 200 samples analysed, a report would be sent to keep the system up to date (shown by the activation of trigger 1). However, it can be seen that on the third and fifth report sent, trigger 2 was activated, indicating that the collaboration rate was very high, causing the agent to react to this major change, making it necessary to evaluate the changes made, as this could quickly degrade the system's efficiency. Furthermore, after trigger 2 was activated, it was noted that the next periodicity trigger maintained the interval of 200 samples.

The choice of the N value is crucial for the system to work properly when sending reports. The choice of this value should not be arbitrary, but should be aligned with the system's needs. These include the cost of communication, as each report sent consumes bandwidth, processing

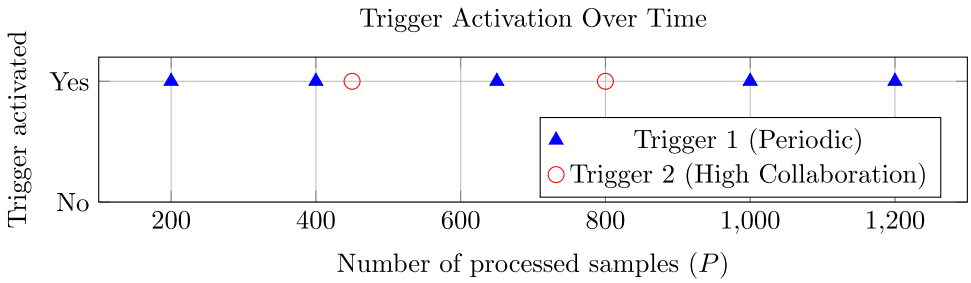


Fig. 5 Example of report triggering based on periodicity and collaboration rate

and time, making it necessary to balance the frequency of communication with the GAs (usually hosted in the cloud), and the computational cost of sending reports, avoiding overloading the IoT networks. It can also be in relation to the size of the statistical window, where an adequate number of samples guarantees significant statistics to represent the system, allowing a good evaluation of the model and the decisions made. On the other hand, the value of N can also be defined considering latency and operational frequency, for example, if the agent analyzes one sample per second, it would take approximately 16 minutes to analyze 1000 samples. This defines the frequency of parameter updates, making re-evaluations at specific time periods.

Although initially defined as a fixed value, the N parameter will be adapted dynamically in the future, based on the agent's behaviour, variations in collaboration and the model's effectiveness. In this way, the system will be able to reduce N at times of instability (wrong answers, too many collaborations) or increase it when confidence in the model is high, optimizing the use of the network and resources.

Reinforcement Learning Model

In this work, RL is adopted as the core mechanism used by GAs to optimize the behaviour of LAs in a decentralized IDS. Rather than relying on fixed rules or static configurations, GAs continuously learn from behavioural reports sent by LAs and suggest tactical parameter adjustments that improve decision quality and system performance over time.

The RL agent does not directly access sensitive data or raw network traffic, but instead learns through indirect feedback provided by LAs, encoded in compact behavioural reports. This setup ensures that the learning process is privacy-preserving while still being capable of capturing key trends in LA performance.

In this architecture, the learning task focuses specifically on optimizing two decision parameters:

- θ: the confidence threshold that determines whether a LA makes a prediction autonomously or requests collaboration from peers;
- α: the weight assigned to the LA's own prediction versus the aggregated responses when collaboration occurs.

Both parameters directly influence the LA's decision-making strategy, particularly its behavior during collaborative inference. For instance, a lower θ encourages collaboration even in moderately uncertain situations, while a higher α prioritizes self-reliance in the final decision. Therefore, adjusting these parameters allows the system to shift between more autonomous or more collaborative behaviours, depending on context.

Learning Strategy

To model the sequential decision-making process of tuning θ and α, it was employed a Q-learning algorithm. Each GA maintains a separate Q-table for each LA under its supervision, capturing the learned utility of taking specific actions (parameter adjustments) in given system states.

The main characteristics of the learning model are as follows:

- **State representation**: Each state corresponds to a discretized combination of (θ, α) values currently in use by the LA.
- **Action space**: The agent can perform small or medium adjustments to θ and/or α, or maintain the current configuration. This allows for fine-grained control of decision dynamics.
- **Reward signal**: After each interaction, a reward is computed based on the agent's recent performance (e.g., accuracy, false positive rate, collaboration rate). This feedback is more immediate than global accuracy and enables quicker adaptation.
- **Exploration policy**: An ϵ-greedy strategy is used to balance exploration and exploitation. Each LA has its own ϵ value that decays over time, allowing personalized convergence to optimal behaviors.
- **Learning update**: Q-values are updated using the standard temporal-difference formula. Only feedback from local intervals is used, ensuring that learning reflects the short-term impact of each action.

This design ensures that the system remains flexible, privacy-preserving, and adaptive to evolving attack patterns or traffic behaviours. Rather than adjusting models themselves, which would be computationally intensive and potentially violate local constraints, the RL agent focuses on influencing decision policies through lightweight but impactful parameter tuning.

Next subsection formally describe the learning problem as a Markov Decision Process (MDP), detailing the mathematical modeling of states, actions, rewards, and the learning algorithm.

Problem Formulation as a Markov Decision Process

The problem of dynamically adjusting the θ and α decision parameters in distributed intrusion detection agents can be modeled as a MDP. The decision-making is based uniquely on the current system state, satisfying the Markov property, which assumes that future states depend only on the current state and the chosen action, and not on the sequence of previous states.

MDP Components

Formally, it was defined the MDP as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the state space. Each state $s \in \mathcal{S}$ is defined by the current values of the decision thresholds (θ, α), which influence the behaviour of LAs when analyzing data samples. This space was discretized for efficient learning.
- $\mathcal{A}$ is the action space. Each action $a \in \mathcal{A}$ represents an adjustment to the current values of θ and/or α. It was considered different magnitudes of changes, both individual and combined, such as:

$$\mathcal{A} = \{(\pm 0.05, 0), (0, \pm 0.05), (\pm 0.1, 0), (0, \pm 0.1)\}$$

including also compound updates, e.g., (0.05, 0.05), $(0.1, -0.1)$.

- $\mathcal{P}$ is the state transition probability function, which is not known a priori and is implicitly learned through interactions with the environment.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. The reward is computed based on several metrics reported by the LA, including:
  - Model accuracy (to be maximized)
  - False positive and false negative rates (to be minimized)
  - Collaboration request rate (to be kept under a desirable threshold)

These components are averaged to generate the scalar reward:

$$R(s,a) = \frac{1}{4}\left(\text{accuracy} + (1 - \text{FP rate}) + (1 - \text{FN rate}) + (1 - \text{collab. rate})\right) \tag{7}$$

- $\gamma \in [0, 1]$ is the discount factor, which balances the importance of immediate versus future rewards.

The main objective of the RL agent is to find the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maximises the expected discounted return (G_t) from any initial state s_0. The return G_t is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{8}$$

The optimal policy is therefore one that satisfies:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[G_t \mid s_t = s_0\right] \tag{9}$$

The discount factor $\gamma \in [0, 1]$ is fundamental in this formulation, as it weights the importance of future rewards relative to immediate rewards, ensuring that the agent makes decisions that are globally optimal throughout its interaction with the environment. In this implementation of Q-Learning, γ is used explicitly in Bellman's Equation to calculate the value estimate Q.

Impact and Functioning of Decision Parameters (θ and α)

The RL module operates by evaluating the utility (Q value) of adjusting θ and α, searching for values that maximize the expected discounted reward. Since the reward $\mathcal{R}$ depends on classification performance and collaboration efficiency, adjusting these parameters directly influences the final decision of the LA.

To illustrate the practical impact of θ and α on an LA's decision, consider a LA that has confidence in its own prediction of $C_{\text{own\_pred}} = 0.60$ for a given data sample, this confidence varies for each sample, being the probability of the prediction for the sample under analysis. Consider also that the LA's own prediction is $\hat{y}_{\text{own}} = 1$ (Attack).

Step 1: Solo versus Collaborative Decision (θ)

The LA uses the confidence threshold θ to decide whether to apply its own prediction (solo decision) or request collaboration (group decision).

- **Case A (collaborative decision)**: Considering $\theta = 0.70$, as the confidence of LA $C_{\text{own\_pred}} = 0.60$ is lower than $\theta = 0.70$, the agent is considered uncertain and should request the collaboration of other LAs present in the system (proceed to step 2).
- **Case B (solo decision)**: Considering $\theta = 0.55$, since the LA's confidence $C_{\text{own\_pred}} = 0.60$ is greater than $\theta = 0.55$, the agent is considered confident and applies its own prediction (e.g., $\hat{y} = 1$ if its own prediction was Attack), ending the decision process without communication.

The RL agent learns that if case B (low θ) leads to higher overall rewards ($\mathcal{R}$), it should favour actions that reduce θ, promoting autonomy and network efficiency.

Step 2: Aggregation Decision (α)

This step is only performed if the LA opts for collaboration (i.e., in case A, where $\theta = 0.70$). The LA receives the responses from other LAs present in the system (e.g., LA_1 to LA_5). In the implementation of this work, confidence (C_i) and weight (w_i) are considered the same metric (the probability of prediction $\hat{y}_i$).

LA (i)	Prediction ($\hat{y}_i$)	Confidence ($C_i = w_i$)
LA_1	1 (Attack)	0.85
LA_2	0 (Normal)	0.95
LA_3	1 (Attack)	0.80
LA_4	0 (Normal)	0.90
LA_5	0 (Normal)	0.65

The LA uses the aggregation threshold α to determine the final classification, after filtering the responses of other agents whose confidence C_i is lower than its own collaboration θ (0.70).

1. **Indicator function action** (Eq. 1): Considering $\theta = 0.70$ from the requesting LA, the indicator function $\mathbb{I}(C_i \geq \theta)$ is applied to all contributions. Only LAs with $C_i \geq 0.70$ are considered: LA_1 ($C_1 = 0.85$), LA_2 ($C_2 = 0.95$), LA_3 ($C_3 = 0.80$), and LA_4 ($C_4 = 0.90$) are reliable. The requesting LA ($C_{\text{own\_pred}} = 0.60$) and LA_5 ($C_5 = 0.65$) are discarded/ignored.
2. **Weighted average (WA):** Only the four reliable peer responses are used. The total weight of reliable contributions is:

$$W_{\text{total}} = C_1 + C_2 + C_3 + C_4 = 0.85 + 0.95 + 0.80 + 0.90 = 3.50$$

The WA for the attack class ($\hat{y} = 1$) is calculated:

$$\begin{aligned} W_{\text{attack}} &= (C_1 \cdot \hat{y}_1) + (C_2 \cdot \hat{y}_2) + (C_3 \cdot \hat{y}_3) + (C_4 \cdot \hat{y}_4) \\ &= (0.85 \cdot 1) + (0.95 \cdot 0) + (0.80 \cdot 1) + (0.90 \cdot 0) = 1.65 \end{aligned}$$

The WA for the attack class is $WA = W_{\text{attack}}/W_{\text{total}} = 1.65/3.50 \approx 0.471$.

3. **Final decision** (α): The final decision $\hat{y}$ is determined by comparing *WA* with α:

- **Case C**: Considering $\alpha = 0.50$ (high consensus), since $WA = 0.471$ is *lower* than $\alpha = 0.50$, the final decision is $\hat{y} = 0$ (Normal).
- **Case D**: Considering $\alpha = 0.45$ (low consensus), since $WA = 0.471$ is *greater* than $\alpha = 0.45$, the final decision is $\hat{y} = 1$ (Attack).

The RL agent will dynamically adjust α, aiming to maximize the subsequent reward $\mathcal{R}$.

In summary, the combination of θ and α will cause values to be dynamically adjusted for the agents' decisions, highlighting the agent's good predictions in cases of high confidence/probability and, on the other hand, correcting erroneous predictions through collaboration. In previous work [6], considering static values for θ and α has already shown promising results, in which it was possible to verify the correction of erroneous predictions, with a correction rate ranging from 7% to 32%.

Justification

This formulation allows the system to learn optimal adaptation policies for θ and α based on the real-time feedback from LAs. The use of RL (specifically, Q-learning) enables the system to continuously improve decision-making under uncertainty and dynamic conditions. The agent converges to a stable configuration by observing the evolution of model accuracy, system collaboration behaviour, and misclassification rates.

Moreover, the incorporation of exploration (via ϵ-greedy strategy) and state discretization helps to ensure that the solution is robust even in the presence of noisy reports or fluctuating traffic patterns.

Experimental Setup and Case Study

This section describes the experimental infrastructure designed to validate the proposed architecture for distributed intrusion detection in IoT environments. The setup emulates a real-world scenario where multiple IoT devices cooperate in the detection of attacks, while a high-level entity monitors and assists the system adaptively.

System Application Layer: Local Agents Running on IoT Devices

The System Application Layer was implemented using Raspberry Pi 3 Model B+ devices, each acting as a distinct edge node in a distributed IoT network. These devices emulate heterogeneous IoT equipment with limited computational capacity and are responsible for performing local intrusion detection. Each Raspberry Pi executes an agent, referred to as an LA, responsible for orchestrating the decision-making process and communication between agents. The structure of these edge devices with collaboration and attack detection capabilities is illustrated in Fig. 6.

To simulate realistic network conditions and device behaviour without requiring live traffic capture, each LA was fed with pre-collected data from the CICIoT2023 dataset [34]. This dataset includes traffic data from 105 real IoT devices, representing various benign and attack scenarios, making it highly suitable for emulating detection in heterogeneous environments.

Each device contains two processes, the LA and the ML engine. The function of the LA is to acquire network data, which in this case is done by reading the data that has already been collected (CICIoT2023 dataset), extracting the main features and sending this data to the second process (ML engine) to predict whether the data is normal or an attack. For this communication, the two processes exchange information via a shared memory in the kernel, through the Unix socket. The GA, represented as the server, contains the RL engine to perform the analysis of reports received by the LAs.

In case the local prediction confidence fell below a predefined threshold (θ), the LA issued a collaboration request to neighboring LAs, aggregating their responses using a weighted average strategy (see Eq. 1) controlled by the parameter α. This strategy allowed agents to combine insights from their peers, potentially improving detection accuracy.

System Support Layer: Reinforcement Learning model

To support adaptive behaviour, the System Support Layer includes a GA deployed on a high-performance computing platform (e.g., a workstation or private cloud server). This agent receives behavioural reports generated by each LA at regular intervals or upon triggering conditions, summarizing

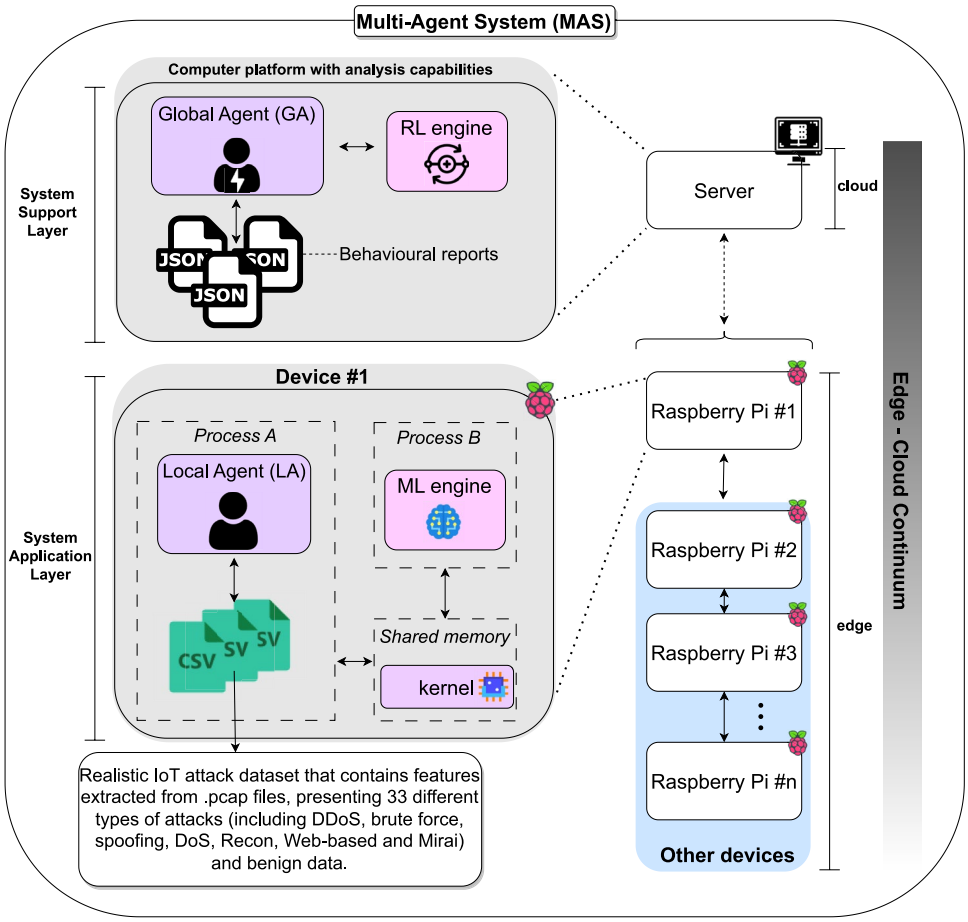


Fig. 6 Structure of edge devices with collaboration and attack detection capabilities in *System Application Layer* and adaptation mechanism in *System Support Layer* (adapted from [6])

local performance metrics such as accuracy, collaboration rate, false positives/negatives, CPU and memory usage.

Each report is structured as a JSON object and includes metrics like decision thresholds (θ, α), collaboration requests, model accuracy, detected attacks, and system resource usage. These reports serve as the observations for a RL agent running within the GA.

The RL module, implemented in Python, models the adaptation process as a MDP and uses Q-learning to suggest updates to the θ and α parameters of each LA. Each LA is assigned an individual Q-table and independent exploration rate (ϵ), allowing personalized adaptation based on local history. The learning policy evolves over time by analyzing rewards derived from the reports, which consider factors such as local detection accuracy, false positives/negatives, and collaboration rate.

Communication and Execution Workflow

The workflow diagram shown in Fig. 7 illustrates the complete communication and execution cycle of the proposed system. The flow highlights the interactions between the LAs and the GA, from the initial analysis of network traffic to the adaptation of decision parameters. This process aims to guarantee a coordinated and efficient response to uncertainties in detection, promoting continuous learning and collaboration between agents.

The complete system workflow can be summarized as follows:

1: Each LA reads network traffic samples from the dataset and performs ML-based analysis.

2: If prediction confidence is low, collaboration with other LAs is initiated (see defined protocol in Fig. 3).

3: After receiving the analysis from other LAs described in the collaboration protocol, or if confidence is high, the LA initiates the decision-making process. If confidence is high, the decision is made independently, and if it is low, it is made collaboratively (based on Eq. 1).

4: Behavioural reports are updated periodically (information described in Table 1).

5: If triggers for sending reports are activated (described in Eqs. 2 to 6), reports are sent to the GA, and LA waits to receive new recommendations.

6: The GA receives the report, processes it, updates the Q tables (see protocol defined in Fig. 4) and suggests new values θ and α to optimize local decision-making.

7: LAs update their parameters based on the recommendations and continue the analysis cycle.

All modules were deployed and tested in a controlled environment to ensure stability and reproducibility. Metrics collected throughout the experiments were logged and later analyzed to assess system performance, convergence, collaboration behaviour, and the overall effectiveness of the RL-based adaptive mechanism.

To ensure the confidentiality and authenticity of interactions, the exchange of messages between LAs and between LAs and GAs is standardized by the FIPA Agent Communication Language (ACL). Furthermore, this communication is protected by a hybrid end-to-end encryption scheme. The message content is encrypted using the symmetric AES algorithm to ensure high speed and efficiency, minimising computational overhead. At the same time, the symmetric key used is protected by the asymmetric RSA algorithm, employing the public key of the recipient agent. The choice of this hybrid approach is dictated by the computational

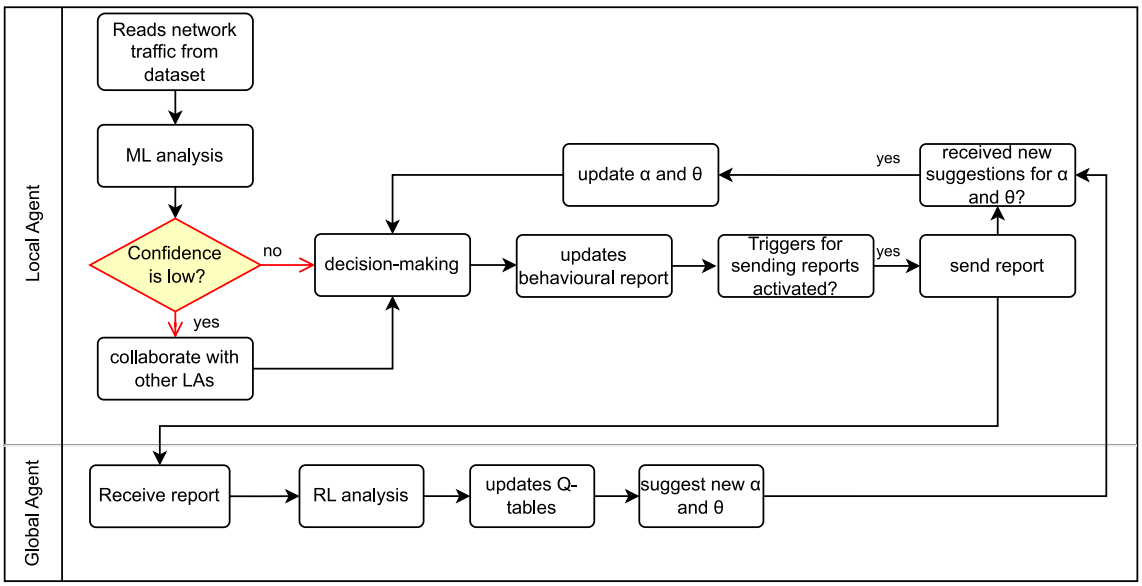


Fig. 7 Communication and execution workflow between LAs and GA in the proposed system

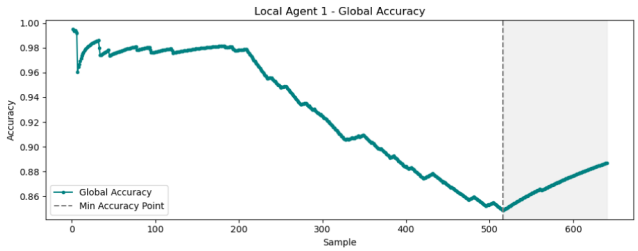
power limitations inherent in IoT devices located at the edge, as well as the need to optimise encryption and decryption performance, which depends directly on the variable sizes of messages exchanged in the edge-cloud continuum.

Experimental Results and Discussion

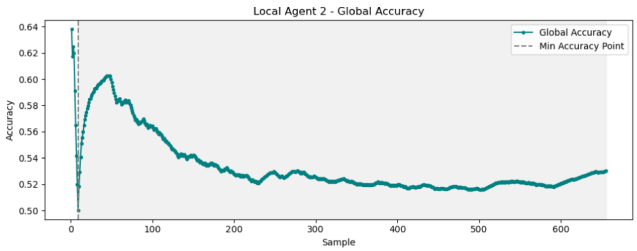
This section presents the experimental results obtained from the proposed MAS architecture, focusing on the impact of dynamic RL-based adjustments in the System Support Layer. The evaluation aims to assess how the continuous optimization of the decision parameters θ and α influences the performance of LAs in terms of accuracy, collaboration behaviour, and anomaly detection efficiency. The analysis includes individual metrics for each LA, comparative behaviours, and evidence of system adaptation and convergence over time. Although the performance of the agent is driven by a composite reward function $\mathcal{R}$ (including acurracy, false positives, false negatives, and collaboration rate), accuracy evolution is used as the primary and visible metric to demonstrate the successful convergence and adaptability of the RL mechanism.

Figure 8 presents the evolution of the global accuracy over time for each LA, where each curve corresponds to one of the distributed agents in the system. The x-axis represents the number of processed samples (or iterations between LA and GA), and the y-axis shows the global accuracy reported in each behavioural report. Additionally, a vertical dashed line marks the point of minimum accuracy, and the shaded area highlights the period after that point, allowing visual assessment of the agents' recovery and stabilization behaviours.

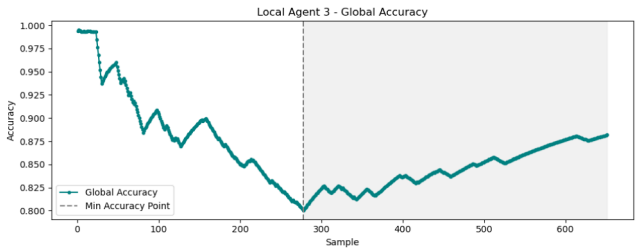
It is important to highlight the initial performance degradation consistently observed across all agents, characterized by a marked drop in accuracy during the early learning phase. This behaviour is a direct consequence of the exploration-driven dynamics inherent to the Q-learning algorithm,



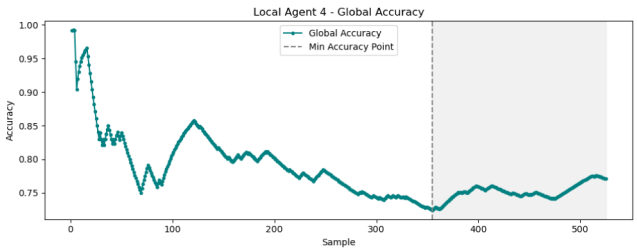
(a) Global accuracy for LA 1 during action exploration.



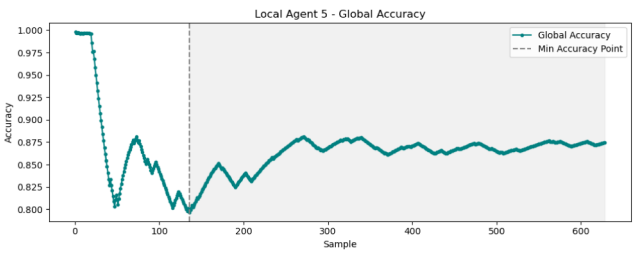
(b) Global accuracy for LA 2 during action exploration.



(c) Global accuracy for LA 3 during action exploration.

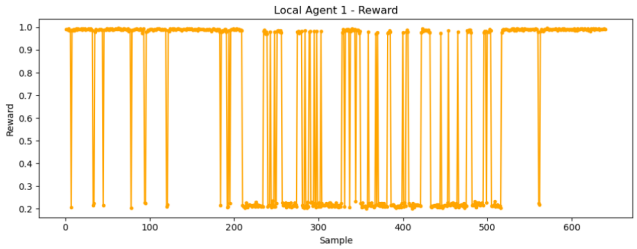


(d) Global accuracy for LA 4 during action exploration.

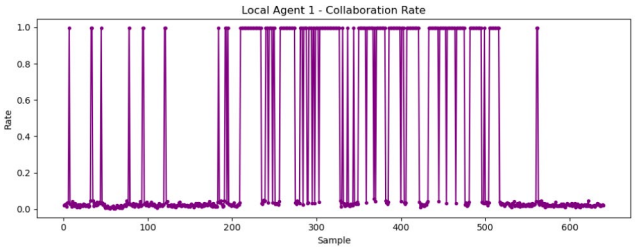


(e) Global accuracy for LA 5 during action exploration.

Fig. 8 Global accuracy for local agents during action exploration



(a) Reward for LA 1 during action exploration.



(b) Collaboration rate for LA 1 during action exploration.

Fig. 9 Reward and collaboration rate for LA 1 during action exploration

wherein elevated ϵ values induce stochastic action selection. As a result, agents systematically explore a wide range of parameter configurations, specifically adjustments to θ and α, to empirically identify optimal strategies that maximize long-term policy performance guided by the composite reward function $\mathcal{R}$. However, after reaching a minimum value of global accuracy (represented by the vertical line on the graph), all agents demonstrate either stabilization or recovery of their accuracy levels.

This behaviour indicates that the agents are starting to exploit previously learned actions with better performance, meaning the Q-table is being updated effectively based on received rewards. It is possible to notice the evidence of local learning, in LAs 1, 3 and 5, there is clear recovery and maintenance of higher accuracy levels after the critical point, suggesting that the RL system is adapting appropriately by tuning the θ and α parameters. Also, it can be observed the individualized behaviour across agents, where each agent demonstrates a distinct trajectory, reinforcing the importance of maintaining separate Q-tables and ϵ values. For example, while LA 1 ends with a global accuracy close to 90%, LA 4 stabilizes around 80%, indicating differences in environment, model, or collaborative dynamics.

It is important to note that the global accuracy metric is cumulative, meaning it reflects the agent's performance over the entire sequence of processed samples. In this way, once the accuracy drops due to a prolonged period of poor decisions (e.g., caused by the exploration of suboptimal parameters), it becomes difficult to return to the initial high accuracy values. Even if the agent subsequently learns and consistently makes correct predictions, the earlier degradation continues to affect the cumulative metric. Therefore, the fact that the global accuracy stops decreasing and remains stable in the final phase, despite not returning to the original level, is a strong indication that the agent has successfully learned an optimal policy (exploitation), is consistently making effective local decisions, confirming the success of the dynamic adjustment mechanism, and is achieving high local accuracy.

Although most of the agents showed good recovery and improved response after exploration and learning, LA 2, which initially didn't show good results (it was intentionally set up this way to observe its behaviour), only began to improve and stabilise after 600 interactions with the GA. It can therefore be concluded that for agents with models that perform poorly, the learning time may be too long.

To enable a more comprehensive analysis of the system's behaviour, a detailed examination is conducted focusing on Local Agent 1 (LA 1) as a representative case. Figure 9 depicts the evolution of two key metrics throughout the experiment: *reward* and *collaboration rate*, which will be analysed together with the overall accuracy shown in Fig. 8a.

Initially, LA 1 starts with a high accuracy, which remains stable for a considerable number of samples. During this early phase, the agent receives high rewards and maintains a low collaboration rate, suggesting that the model in use was adequate and confident in its predictions, requesting help only in a few uncertain cases.

However, as the RL algorithm explores different values of decision parameters θ and α, a gradual degradation in global accuracy can be observed. This probably results from the selection of suboptimal parameters that reduced the agent's ability to make accurate decisions independently. Consequently, the agent increases its reliance on peer collaboration to compensate for local model uncertainty. This behaviour is reflected in the peak in the collaboration rate and a notable drop in the rewards received.

Over time, the agent adapts to this feedback, and after reaching its lowest point of accuracy, it begins to recover and stabilize. As the Q-learning process favors actions that result in better performance, the agent learns to select more appropriate values for θ and α, reducing its dependency on external collaboration. This learning process leads to a consistent improvement in both reward and decision quality, as evidenced by the decrease in collaboration requests and the rise in reward values during the final phase.

This observed pattern is consistent with the behavior of other agents in the system, confirming that the proposed

RL strategy enables each agent to adapt over time, recover from performance degradation caused by exploration, and converge toward more optimal decision-making configurations. These results demonstrate the effectiveness of individualized RL-based parameter adjustment in maintaining or improving detection accuracy within a distributed and dynamic environment.

Conclusions and Future Work

The exponential growth of connected devices and the increasing sophistication of threats in IoT environments demand security mechanisms that are not only innovative but also adaptive and privacy-preserving. This work addressed this challenge by proposing a collaborative and decentralized MAS designed to distribute intelligence across IoT networks. A central focus was placed on empowering local agents to operate autonomously at the edge, supported by a higher-level adaptive layer that enhances decision-making without compromising data privacy.

An important gap identified in the current literature is the limited integration of distributed intelligence into the edge-cloud continuum, especially in architectures that simultaneously leverage the proximity and responsiveness of edge computing with the computational power and global perspective of cloud-based coordination. While previous work has explored MAS and collaboration strategies, few works have effectively combined real-time behavioural feedback with adaptive learning mechanisms that operate cohesively across this continuum. The focus of existing approaches is on centralised solutions that compromise latency and privacy or rely only on edge-based intelligence, which may lack the adaptability and scalability required in dynamic environments. The proposed system addresses this gap by distributing decision-making across both layers, edge and cloud, capitalising on the strengths of each, i.e. low latency, privacy-preserving local analysis at the edge and high-level adaptive coordination in the cloud.

To address this, it was introduced a novel architecture composed of two synergistic layers: the System Application Layer, where distributed local agents operate on resource-constrained edge devices, and the System Support Layer, where more powerful agents employ reinforcement learning to monitor, guide, and optimize system behaviour. A distinctive feature of this architecture is its use of behavioural reports, containing operational indicators such as accuracy, collaboration rate, and resource usage, as real-time feedback to dynamically adjust decision parameters used in local agents. This enables continuous improvement in decision-making at the edge without requiring access to raw or sensitive data, thereby preserving privacy.

Experimental results demonstrated that even with a basic Q-learning implementation, the system was capable of learning, adapting, and enhancing agent behaviour over time. It effectively reduced collaboration overhead and improved decision accuracy, while also exhibiting resilience by recovering from initial performance degradation due to exploration. These findings confirm the robustness and adaptability of the proposed approach.

In general, leveraging real-time behavioural feedback for adaptive decision-making has proven to be an effective and scalable strategy for intelligent, distributed security in IoT networks. The proposed architecture offers a solid foundation for building privacy-aware, efficient, and self-improving MAS suitable for IIoT and IoT applications.

Future work will explore several promising directions, namely the integration of predictive models to dynamically adjust the reporting interval, enabling more responsive communication strategies based on system dynamics, the implemention of automated model replacement mechanisms, allowing the support layer to detect and replace underperforming models with more effective alternatives. Also, it will be investigated advanced reinforcement learning techniques, such as Deep Q-Networks (DQN), to enhance scalability and adaptability in more complex and nonlinear environments.

Author Contributions Gustavo Funchal was responsible for the implementation, experimentation, and writing of the manuscript. The core ideas and conceptual foundation of the work were developed collaboratively, with significant contributions from Paulo Leitão, Tiago Pedrosa, and Fernando de La Prieta, who also provided continuous guidance throughout the research process and reviewed the manuscript to improve its quality. All authors read and approved the final version of the manuscript.

Funding Open access funding provided by FCT|FCCN (b-on). This work has been supported by national funds through FCT/MCTES (PIDDAC): CeDRI, UIDB/05757/2020 (DOI: 10.54499/UIDB/05757/2020) and UIDP/05757/2020 (DOI: 10.54499/UIDP/05757/2020); and SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020). This work has been also supported and received funding from "CyberPRAISE - Cybersecurity research for PRivAte, Intelligent and truStablE solutions" - NORTE2030-FEDER-01820300. The author Gustavo Funchal thanks the FCT Portugal for the PhD Grant 2022.13712.BD.

Data Availability The dataset used in this study is publicly available at the following URL (https://www.unb.ca/cic/datasets/iotdataset-2023.html).

Declarations

Conflict of interest We hereby declare that we have no conflict of interest to disclose, whether financial or non-financial, that are directly or indirectly related to the work presented in this paper. We confirm that there are no personal, professional, or commercial affiliations that may influence or bias the content of this manuscript. This declaration is made to ensure transparency and integrity in the publication process.

Informed Consent Not applicable.

Research Involving Human and/or Animals Not applicable.

References

1. Morenas J, Silva CM, Funchal GS, Melo V, Vallim M, Leitao P. Security experiences in iot based applications for building and factory automation. in 2020 IEEE International Conference on Industrial Technology (ICIT), pp. 322–7 (2020). https://doi.org/10.1109/ICIT45562.2020.9067229
2. Cziva R, Pezaros DP. Container network functions bringing: nfv to the network edge. IEEE Commun Mag. 2017;55(6):24–31. https://doi.org/10.1109/MCOM.2017.1601039.
3. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: vision and challenges. IEEE Internet Things J. 2016;3(5):637–46. https://doi.org/10.1109/JIOT.2016.2579198.
4. Hamouda D, Ferrag M.A, Benhamida N, Seridi H. Intrusion detection systems for industrial internet of things: a survey. in 2021 ICTAACS (2021). https://doi.org/10.1109/ICTAACS53298.2021.9715177
5. Funchal G, Pedrosa T, Prieta FDL, Leitao P. Edge multi-agent intrusion detection system architecture for IoT devices with cloud continuum. in 2024 IEEE 7th ICPS (2024). https://doi.org/10.1109/ICPS59941.2024.10639952
6. Funchal G, Pedrosa T, Prieta F, Leitão P. Distributed machine learning and multi-agent systems for enhanced attack detection and resilience in iot networks. in Proceedings of the 11th international conference on information systems security and privacy, pp. 192–203. SCITEPRESS - Science and Technology Publications, Porto, Portugal (2025)
7. Alsboui T, Qin Y, Hill R, Al-Aqrabi H. Distributed intelligence in the internet of things: challenges and opportunities. SN Comput Sci. **2**(4) (2021). https://doi.org/10.1007/s42979-021-00677-7
8. Kong L, Tan J, Huang J, Chen G, Wang S, Jin X, et al. Edge-computing-driven Internet of Things A Survey. ACM Computing Surveys 2022;55(8):1–41. https://doi.org/10.1145/3555308.
9. Gyamfi E, Jurcut A. Intrusion detection in internet of things systems: a review on design approaches leveraging multi-access edge computing, machine learning, and datasets. Sensors. 2022;22(10):3744. https://doi.org/10.3390/s22103744.
10. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: vision and challenges. IEEE Internet Things J. 2016;3(5):637–46. https://doi.org/10.1109/JIOT.2016.2579198.
11. Quy NM, Ngoc LA, Ban NT, Hau NV, Quy VK. Edge computing for real-time Internet of Things applications: future internet revolution. Wireless Pers Commun. 2023;132(2):1423–52.
12. Ghosh S, Mukherjee A, Ghosh SK, Buyya R. Mobi-IoST: mobility-aware cloud-fog-edge-iot collaborative framework for time-critical applications. IEEE Trans Netw Sci Eng. 2020;7(4):2271–85. https://doi.org/10.1109/TNSE.2019.2941754.
13. Wang H, Liu T, Kim B, Lin C, Shiraishi S, Xie JL, Han Z. Architectural design alternatives based on cloud/edge/fog computing for connected vehicles. CoRR **abs/2009.12509** (2020)
14. Xie R, Tang Q, Wang Q, Liu X, Yu FR, Huang T. Collaborative vehicular edge computing networks: architecture design and research challenges. IEEE Access. 2019;7:178942–52. https://doi.org/10.1109/ACCESS.2019.2957749.
15. Qadir J, Sainz-De-Abajo B, Khan A, García-Zapirain B, De La Torre-Díez I, Mahmood H. Towards mobile edge computing: taxonomy, challenges, applications and future realms. IEEE Access **8**, 189129–189162 (2020). https://doi.org/10.1109/ACCESS.2020.3026938
16. IEEE Computer Society: Cloud Continuum, Volume 2, Number 1 (2022)
17. Dell Technologies: Edge to Cloud Continuum Overview. https://infohub.delltechnologies.com/it-it/l/integrated-edge-management-in-smart-manufacturing-white-paper-1/edge-to-cloud-continuum-overview/. White paper (2022)
18. Duan J, Zhang S, Wang Z, Jiang L, Qu W, Hu Q, Wang G, Weng Q, Yan H, Zhang X, Qiu X, Lin D, Wen Y, Jin X, Zhang T, Sun P. Efficient training of large language models on distributed infrastructures: a survey. arXiv (2024). https://doi.org/10.48550/ARXIV.2407.20018
19. Khouas A.R, Bouadjenek M.R, Hacid H, Aryal S. Training machine learning models at the edge: a survey (2024). arXiv:2403.02619
20. Mwase C, Jin Y, Westerlund T, Tenhunen H, Zou Z. Communication-efficient distributed AI strategies for the IoT edge. Future Gener Comput Syst. 2022;131:292–308. https://doi.org/10.1016/j.future.2022.01.013.
21. Wang YE, Wu C-J, Wang X, Hazelwood K, Brooks D. Exploiting parallelism opportunities with deep learning frameworks (2020). arXiv:1908.04705
22. Jia Z, Zaharia M, Aiken A. Beyond data and model parallelism for deep neural networks (2018). arXiv:1807.05358
23. Mirhoseini A, Pham H, Le Q.V, Steiner B, Larsen R, Zhou Y, Kumar N, Norouzi M, Bengio S, Dean J. Device placement optimization with reinforcement learning (2017). arXiv:1706.04972
24. Zhou Z-H. Ensemble learning, pp. 181–210. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-1967-3_8
25. Prazeres N, Costa RL, Santos L, Rabadão C. Engineering the application of machine learning in an IDS based on IoT traffic flow. Intell Syst Appl. 2023;17:200189. https://doi.org/10.1016/j.iswa.2023.200189.
26. Berman DS, Buczak AL, Chavis JS, Corbett CL. A survey of deep learning methods for cyber security. Information 10(4) (2019). https://doi.org/10.3390/info10040122
27. Mohy-eddine M, Guezzaz A, Benkirane S, Azrour M. An effective intrusion detection approach based on ensemble learning for IIoT edge computing. J Comput Virol Hack Tech. 2022;19(4):469–81. https://doi.org/10.1007/s11416-022-00456-9.
28. Yang R, He H, Xu Y, Xin B, Wang Y, Qu Y, et al. Efficient intrusion detection toward IoT networks using cloud-edge collaboration. Comput Netw. 2023;228:109724. https://doi.org/10.1016/j.comnet.2023.109724.
29. Wooldridge M. An introduction to MultiAgent systems. Wiley, Chichester, UK (2009). https://books.google.pt/books?id=X3ZQ7yeDn2IC
30. Queiroz J, Leitão P, Barbosa J, Oliveira E. Distributing intelligence among cloud, fog and edge in industrial cyber-physical systems. in Proceedings of the 16th ICINCO. SCITEPRESS - Science and Technology Publications, Prague, Czech Republic (2019). https://doi.org/10.5220/0007979404470454

31. Leitão P. Agent-based distributed manufacturing control: a state-of-the-art survey. Eng Appl Artif Intell. 2009;22(7):979–91. https://doi.org/10.1016/j.engappai.2008.09.005.
32. Leitão P, Colombo AW, Karnouskos S. Industrial automation based on cyber-physical systems technologies: prototype implementations and challenges. Comput Ind. 2016;81:11–25. https://doi.org/10.1016/j.compind.2015.08.004.
33. Ali N, Aloi G, De Rango F, Savaglio C, Gravina R. Edge-cloud continuum driven industry 4.0. Procedia Comput Sci. 2025;253:2586–94. https://doi.org/10.1016/j.procs.2025.01.318.
34. Neto ECP, Dadkhah S, Ferreira R, Zohourian A, Lu R, Ghorbani AA. CICIoT2023: a real-time dataset and benchmark for large-scale attacks in IoT environment. Sensors **23**(13) (2023). https://doi.org/10.3390/s23135941