Resilient Wireless Sensor Actor Networks Through Multi-Objective Self-Adaptation

Ruben Gomes and Noélia Correia

***Abstract*—Wireless Sensor Actor Networks (WSAN) are a key enabler of Internet of Things applications that demand timely and reliable data exchange under dynamic conditions. Among the various domains that benefit from these networks, precision agriculture stands out, demanding adaptive strategies for effective monitoring and control. This study proposes a reinforcement learning approach that leverages the Operationalization construct of the Self-Orchestrated Web of Things (SOrWoT) framework to enhance the adaptability of Things' internal operations. A problem is formulated as a Markov Decision Process, and a Deep Q-Learning agent is trained in a custom simulation environment to identify the most suitable Operationalizations for optimizing data accuracy and latency, under changing conditions and communication failures. The results show that during normal operation the agent favored parallel sensor data averaging to minimize read error, but after an actor failure and the consequent increase in sensor-to-actor distances, it adapted by prioritizing latency through faster Operationalization choices. Sensitivity analyses further confirmed the agent's ability to adjust policies in response to partial failures, and to shifts in the relative importance of latency versus accuracy. These findings demonstrate that reinforcement learning can autonomously optimize WSAN performance, contributing to resilient and self-adaptive systems.**

***Index Terms*—Internet of Things, wireless sensor and actor networks, optimization, deep reinforcement learning.**

I. Introduction

THE advent of the Internet of Things (IoT) has transformed modern industries by introducing networks of interconnected devices capable of collecting, exchanging, and analyzing data autonomously [1]. Building upon this, the Web of Things (WoT) extends interoperability across heterogeneous devices through standard web protocols such as the constrained application protocol (CoAP), promoting scalable and seamless integration in large distributed systems [2], [3].

Among the various domains benefiting from these advances, precision agriculture has emerged as one of the most prominent applications of the IoT paradigm [4], [5], [6]. By deploying connected sensors, drones, and automated machinery, farmers can now monitor soil conditions, weather dynamics, and crop health in real time, enabling data-driven decisions that improve productivity and resource efficiency [7], [8], [9], [10].

At the core of the aforementioned intelligent agricultural systems are wireless sensor actor networks (WSANs) and communication protocols such as Zigbee. WSANs form the communication and control backbone of IoT-enabled farming [11], consisting of spatially distributed sensor nodes that collect environmental data, and actor nodes that process data and perform control actions.

In this context, the concept of an *actor* extends beyond a simple *actuator*: an actor integrates sensing, actuation, and embedded intelligence to autonomously make local decisions based on data analytics or learning models [12]. This embedded intelligence is the key enabler of self-adaptation and optimization in a WSAN environment, allowing actors to dynamically adjust their behavior in response to changing conditions. Nevertheless, optimizing how these intelligent actors interact with their associated sensors under variable environmental and network conditions remains a significant challenge.

To address this, frameworks such as the Self-Orchestrated Web of Things (SOrWoT) provide a means to dynamically adapt the internal operation of Things, by leveraging different Operationalizations according to environmental or performance constraints [13]. Such adaptability is particularly relevant for WSANs in precision agriculture, where environmental variability, communication latency, and potential node failures demand resilient and self-optimizing behavior.

In this study, we leverage Deep Q-Learning (DQL) to train an intelligent agent capable of selecting the most suitable Operationalization type, in order to optimize data accuracy and latency in a WSAN deployed for precision agriculture. The proposed model also accounts for dynamic network changes, including actor node communication failures, allowing the system to reconfigure itself autonomously and maintain operational efficiency.

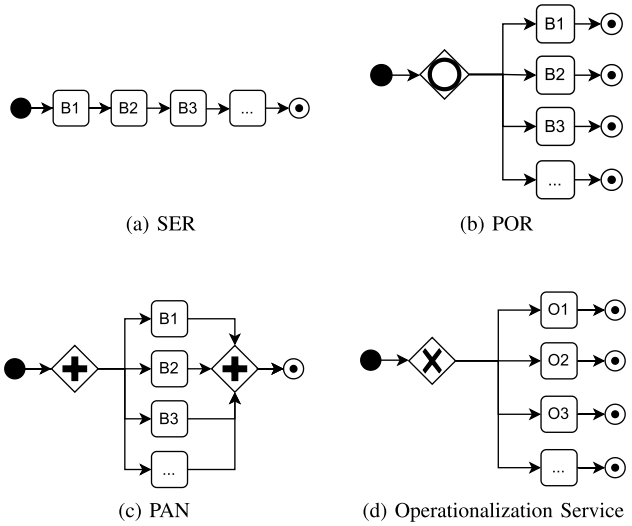
The main contributions of this work are as follows:

- A conceptual mapping of the SOrWoT's framework constructs to the elements of WSANs;
- A Markov Decision Process (MDP) model for optimizing sensor data accuracy and latency by choosing the most suitable Operationalization type;

Received 2 May 2025; revised 21 October 2025 and 26 November 2025; accepted 14 January 2026. Date of publication 20 January 2026; date of current version 29 January 2026. This work was supported by Fundação para a Ciência e Tecnologia (FCT) through the Center for Electronics, Optoelectronics, and Telecommunications (CEOT) under projects UIDB/00631/2020 CEOT BASE and UIDP/00631/2020 CEOT PROGRAMÁTICO, and grant UI/BD/152864/2022. The associate editor coordinating the review of this article and approving it for publication was Z. Xiong. *(Corresponding author: Ruben Gomes.)*

The authors are with the Center for Electronics, Optoelectronics and Telecommunications (CEOT), Faculty of Sciences and Technology, Universidade do Algarve, 8005-139 Faro, Portugal (e-mail: rdgomes@ualg.pt; ncorreia@ualg.pt).

Digital Object Identifier 10.1109/TCCN.2026.3656393



(a) SER (b) POR

(c) PAN (d) Operationalization Service

Fig. 1. Execution across Operationalization types (SER, POR, PAN), and Operationalization Service. B = Behavior, O = Operationalization.

- A DQL agent trained to optimize WSAN performance while dynamically adapting to environmental conditions, including variable noise, latency, and actor failures;
- A custom simulation environment and algorithm for training the DQL agent;
- In-depth analysis of agent optimization performance.

A. Self-Orchestrated Web of Things

The SOrWoT framework suggests using hierarchical finite state machine (HFSM) principles to model the behavior of Things, which are defined by properties, actions and events (i.e. interaction affordances), in the Thing Description information model [14]. The framework, proposed in [13], defines several constructs such as: *Behavior*, with its states and transitions modeled as an HFSM; *Operationalization*, an arrangement of Behaviors; *Operationalization Service*, a set of alternative Operationalizations that can implement an interaction affordance; *Functionality*, a set of interaction affordances used for consumer interaction; and the *Controller*, for coordinating Operationalization Services.

Due to their direct role in the execution of Behaviors, the concepts of Operationalization and Operationalization Service are of special interest to this work.

1) Operationalization: Refers to an execution flow. An Operationalization is an architecting combining one or more Operationalization fundamental types, i.e. its building blocks. The three block types defined in SOrWoT framework are *Serial*, *Parallel OR* and *Parallel AND*, as detailed below:

a) Serial (SER): This type arranges Behaviors in sequential order, meaning each Behavior must complete before the next one can start (see Figure 1a).

b) Parallel or (POR): This type follows a concurrent architecture, meaning its Behaviors are executed simultaneously, and finish independently. Its execution is complete as soon as one Behavior finishes (see Figure 1b).

c) Parallel and (PAN): This type also executes Behaviors concurrently, but requires all Behaviors to finish before allowing execution to proceed (see Figure 1c).

2) Operationalization Service: Contains one or more Operationalizations, and defines a relation of mutual exclusivity among them, meaning only one can be active, i.e. executing, at any given time (see Figure 1d).

B. Motivation and Innovation

The flexibility provided by SOrWoT's Operationalizations can be explored for efficiency gains in systems modeled using such approach. Exploring the impact that Operationalization fundamental types can have in optimizing WSAN systems is an important milestone in the path to develop optimized internal application architectings. For this purpose, we study the use of Operationalization types to enhance both sensor data accuracy and latency, recognizing these metrics' crucial role in precision agriculture, and the trade-off between them. More clearly, we consider that variations may exist in sensors' data accuracy and latency. Such variations may be introduced naturally by sensor reading noise, when considering accuracy, or by communications interference and distance factors, when focusing on latency. This makes the case for an optimization opportunity, with the emergent need to decide which data should be considered and how it should be processed among the available information sources, i.e. the sensors. The term data accuracy, in this context, refers to considering data from more or less sensors, for a more or less accurate result, respectively. It becomes apparent that developing a DQL agent that can learn how to decide which Operationalization type is the most adequate under variable environment conditions, would be desirable.

Generally, the properties of each Operationalization type make them suitable for different situations. The SER type is most useful when latency is not critical, or historical trends need to be gathered [15]. It is characterized by temporal separation, meaning that since data from all sensors is gathered sequentially over a time window rather than a snapshot, temporal fluctuations can be detected (e.g.: area of interest's warming or cooling).

Alternatively, POR should be chosen when fast decisions matter more than data accuracy, delivering advantages such as [16]:

- Comparably Low Latency: Provides data for decision-making, but with a focus on making it available as quickly as possible;
- Resilience to Sensor Failures: Even if some sensors are slow or down, as long as one responds quickly, the actor can proceed.

Lastly, PAN is most useful when data accuracy is crucial, even at the cost of time. It may be selected for the following reasons [17]:

- Statistical Noise Reduction: Aggregating all sensors' readings allows averaging, which minimizes sensor-specific errors and noise;
- Spatial Awareness: Offers a full profile of the area of interest;
- Supports Smarter Decision-Making: Enables subtler actions, according to calculations like averages, that consider multiple sensors to achieve more accuracy.

In precision agriculture, the trade-off between data accuracy and latency has direct implications for decision-making efficiency and resource utilization. High accuracy enables more informed actions such as correctly identifying the precise area or limits that need to be sprayed with a chemical treatment, ensuring that healthy plants are not unnecessarily exposed and resources are used efficiently. Conversely, prioritizing low latency may accelerate decisions—critical when a drone must actuate spraying while in movement—but risks relying on incomplete or noisy data, potentially leading to inefficient resource use. The need for developing decision-support systems in wireless sensor networks (WSNs) has been highlighted in previous works [8], [18].

Existing research has explored WSAN optimization mainly under static assumptions or focused on single-objective improvements [19], [20], [21], but few approaches dynamically adapt to fluctuating network conditions. In particular, actor failures—caused by hardware malfunctions, communication disruptions, or energy depletion—remain insufficiently addressed in current WSAN research [22]. Machine learning offers a promising avenue to explore, where DQLs has demonstrated great potential in optimizing and automating network services through reinforcement learning [23].

To the best of our knowledge, no prior research has explored the use of SOrWoT Operationalization types in a multi-objective optimization of WSANs, automated by a DQL agent capable of adapting dynamically to such failures.

C. Research Goals

This work is guided by the following research question:

- **RQ**: How can a DQL agent adaptively balance multiple optimization objectives in a WSAN environment, using the flexibility of the SOrWoT framework's Operationalizations to adjust its behavior, and ensure minimal disruption during actor failures?

The following sections are organized as follows. Section II describes research works related to this study. In Section III the architecture of WSANs is explained and mapped against SOrWoT's constructs. The WSAN optimization environment is explained in Section IV, while a suitable MDP model is developed in Section V. Section VI introduces DQL and the Deep Q-Network (DQN) model used. The developed simulation's logic and parameters can be found in Section VII, while results are analyzed in Section VIII. Finally, conclusions and future work directions are stated in Section IX.

II. Related Work

WSANs have been the focus of extensive research due to their potential to enable autonomous and resilient cyber-physical systems. Several studies have investigated optimization techniques for enhancing their efficiency, and these are discussed next.

A. Self-Orchestrated Web of Things

The SOrWoT framework was recently proposed to address the adaptability needs of WoT applications by using HFSM constructs to model the behavior of Things [13]. Its core innovation lies in defining modular constructs, such as Behaviors and Operationalizations, that enable dynamic adaptation of a Thing Description's interaction affordances. In the original work, SOrWoT was demonstrated in a smart city energy management scenario, showcasing how Operationalizations can improve resource efficiency and resilience in distributed systems. Beyond this initial application, SOrWoT has not yet been systematically explored in other domains. While HFSMs have been successfully applied in areas such as robotics [24], [25], the integration of HFSM-based Operationalizations into WoT has not yet been addressed. According to the available literature, no prior research has investigated SOrWoT in the context of WSANs or precision agriculture. This paper therefore extends the framework by mapping its constructs to WSAN elements and leveraging reinforcement learning (RL) to optimize sensor data accuracy and latency under dynamic conditions.

B. Reinforcement Learning in WSANs

Reinforcement learning, in particular DQL, is gaining recognition as a method for optimizing decision-making in WSANs. In [26], a performance comparison of RL-based duty cycling techniques was presented with the goal of achieving better energy conservation in IEEE 802.15.4 low-power networks, and this study also highlighted the incapability of sensor nodes running complex energy-costly learning algorithms. Similarly, the work in [27] applied DQL to optimize mobile actors' task scheduling and power control, in order to maximize long-term network data transmission. This was done by dynamically adapting to real-time network conditions and actor positions, providing opportunistic assistance in WSNs. In [28] a simulation system was developed to address the physical positioning of actors in response to events, according to their distributions, using DQNs. In our study, the change in actor position is viewed as affecting the RL agent's environment, to which the agent needs to adapt the current operationalization. This is opposed to considering this change an action that the agent must learn to perform.

C. Data Collection Strategies in WSANs

Sensor data collection and real-time access has been highlighted in [29] as an important factor for allowing predicting trends and decision-making. The authors proposed a three-layered architecture where sensors use low-cost radio-frequency modules to form a Zigbee mesh network, and send gathered data to a coordinator node, which aggregates data and forwards it periodically to a base station where data becomes available via web services. When compared with this work, one can say that our study is more particularly focused on the perception and edge layers, attempting to improve data collection and delay closer to sensors.

Other works, such as [30] propose managing sensor data access through a sensor-cloud, where digital twins of physical sensors are used to provide sensing-as-a-service. In their proposed model, applications request a specific latency requirement which ultimately selects which sensors are used.

In our work, latency is explored as a metric to balance against data accuracy, as opposed to defining particular limits.

D. Actor Mobility and Fault Recovery in WSANs

Fault tolerance in WSANs often requires actor mobility, where actors relocate to maintain network connectivity or assume the role of failed nodes. Previous works such as [31] have explored mobility models for repositioning actors to provide connectivity restoration in disjoint segments of WSNs, while authors in [32] approached a similar problem, though using a controller switching mechanism to ensure operation resilience in case the primary controller failed. In our work, actor mobility is used as well, in order to restore contact with sensors and recover the WSAN's operation, although with the multi-objective of optimizing data accuracy and latency.

E. Multi-Objective Optimization in WSANs

Balancing multiple conflicting objectives is a key challenge in WSANs. The work in [33] used multi-objective optimization, recurring to energy and temperature parameters as a way to configure the sensor field deployment. In [34], an attempt was made at optimizing the conflicting objectives of fault tolerance and communications latency, leveraging virtualization for that purpose. Our work, on the other hand, addresses fault tolerance by demonstrating the adaptability of the reinforcement learning agent, as required after a fault event. Though focusing on sensor network design, the work in [35] highlights the relation between higher accuracies generally incurring on higher measurement costs. The authors optimized the multi-objective goal of minimizing measurement cost while maximizing accuracy recurring to genetic algorithms. Authors in [36] followed a similar optimization approach, while focusing on the problem of improving energy efficiency. They defend that limiting the transmission and reception of data is not enough to decrease energy costs, and that specific solutions need to be developed. They filled this gap by proposing a lossy compression algorithm as a trade-off for energy saving. A trade-off against loss of data accuracy is considered in our study, although in competition with latency improvement, and prioritized depending on the current environment conditions.

A common feature of these multi-objective optimization works is their reliance on genetic algorithms. In our work, we instead use deep reinforcement learning, specifically DQL, which given its trial-and-error approach is particularly well suited to dynamic and evolving environments such as WSANs with actor failures or variable latency requirements. Whereas generic algorithms are often effective for static optimization problems, DQL enables continuous adaptation to changing conditions. This distinction supports our choice of DQL as a more appropriate approach for the dynamic WSAN environment under study.

F. Application Domains of Multi-Objective Optimization

Recent advances in multi-objective optimization have produced a range of biologically inspired and hybrid metaheuristics applied to domains such as healthcare, IoT, and social networks. The NSICA algorithm uses imperialist competition for feature selection in arrhythmia diagnosis [37], while the MOGNDO algorithm is an advancement over the generalized normal distribution optimization, adapted for general multi-objective optimization tasks [38]. Another general purpose algorithm is MOAVOA, which introduces a novel artificial vultures strategy for balancing convergence and diversity in general optimization tasks [39]. In social network analysis, a hybrid multi-objective algorithm combines the Slime Mould and Sine Cosine algorithms to improve overlapping community detection [40], and the AMHS algorithm is introduced in [41] which applies an archive-based harmony search designed to enhance solution diversity and convergence. Lastly, for enhancing botnet detection in IoT environments under conflicting constraints, the work in [42] proposes a dynamic mutation-based Harris Hawks Optimization.

While these methods show strong performance, they rely on population-based heuristics and are designed for relatively static environments. In contrast, our approach formulates WSAN optimization as an MDP and trains a DQN agent to adaptively balance different metrics by learning from interaction with the environment. Also, by leveraging operationalization types from the SOrWoT framework, our method introduces semantic orchestration and resilience not addressed by the above metaheuristics.

To synthesize these diverse contributions and highlight the distinctions between the various research works presented, a unified comparison is provided in Table I. This structured overview clarifies the positioning of our work and underscores the research gap it addresses.

III. Wireless Sensor Actor Network

In a typical architecture of a WSAN, sensor nodes gather data about their environment and communicate it to the actor node associated with the event area the sensor is within. The event area is the physical area of interest of the actor. Packets carrying collected data can either be sent directly to the actor, in a sensor-actor interaction, or indirectly by forwarding the packets through other close-by sensors until the destination actor is reached, in a sensor-sensor coordination [43] (see Figure 2). The WSAN elements play different roles as follows:

- **Server**: Centralized data processing, long-term storage, advanced analytics;
- **Sink**: Intermediary between actors and server, provides data processing, aggregation and short-term storage;
- **Actor**: Handles more advanced processing tasks than sensors, are positioned closer to the data source, can be static or mobile, and include devices such as edge gateways or routers;
- **Sensor**: Responsible for data collection and initial processing. Involves raw data gathering and simple computations. Example devices include location and position sensors, pressure sensors, temperature and humidity sensors among other types.

WSAN elements can be mapped to the SOrWoT framework constructs, and to the different computing layers of perception, edge, fog and cloud, as illustrated in Figure 3. This mapping

TABLE I
SUMMARY OF RELATED IOT RESEARCH WORKS

Ref.	Theme	Approach	Domain / Application	Key Features
[26]	Duty Cycling	Reinforcement Learning	LR-WPAN	Adaptive scheduling; energy efficiency
[27]	Transmission Policy Optimization	Deep Reinforcement Learning	UAV-aided WSN	Opportunistic routing; data optimization
[28]	Actor Mobility	Deep Q-Network	WSAN 3D environment	Actor mobility control; event distribution handling
[29]	Extensible Architecture	Test bed	WSN	Cloud-sensor integration; Real-time data analysis
[30]	Latency-aware Data Management	Sensor-cloud interaction modeling	Latency-sensitive WSN	Cloud-sensor integration, latency guarantees
[31]	Actor Fault Tolerance	DBCE Durability-Based Connectivity Establishment	WSN disjoint segments	Actor mobility; durability-based connectivity
[32]	Control Fault Recovery Fault Recovery	Hardware-in-the-loop test bed	Industrial WSAN	Fast controller switching; fault-tolerance
[37]	Feature Selection and Medical Diagnosis	NSICA Imperialist Competitive Algorithm	Arrhythmia diagnosis	Feature selection; medical data classification
[38]	Multi-objective Optimization	MOGNDO Distribution-based metaheuristic	General multi-objective problems	Novel distribution modeling; benchmark comparisons
[39]	Multi-objective Optimization	MOAVOA Artificial Vultures Optimization	General multi-objective problems	Novel bio-inspired strategy; convergence-diversity balance
[40]	Community Detection in Social Networks	Hybrid SMA-SCA Slime Mould, Sine Cosine Algorithm	Social network overlap community detection	Hybrid strategy; multi-objective clustering
[41]	Multi-objective Optimization	AMHS Harmony Search (archive-based)	General multi-objective optimization	Archive mechanism; diversity preservation
[42]	IoT Security / Botnet Detection	Dynamic Harris Hawks Optimization (mutation-based)	Botnet detection in IoT	Dynamic adaptation; mutation operator
[33]	Sensor Field Configuration	Statistical mechanics	Surveillance networks	Target tracking; probabilistic model
[34]	Fault-tolerant Virtualization	A-NSGA Adapted Non-dominated Sorting Genetic Algorithm	Heterogeneous WSN	Virtualization; fault-tolerance
[35]	Sensor Network Design	NSGA-II Non-Dominated Sorting Genetic Algorithm II	Multirate systems	Sampling rate coordination; multirate extension of Kalman filter
[36]	Data Compression	Evolutionary optimization	WSN	Lossy-aware data compression
this work	Resilient and Self-Adaptive Optimization	DQN-based SOrWoT operationalization adaptation	Unreliable dynamic WSAN	Accuracy-Latency trade-off optimization via dynamic operationalization

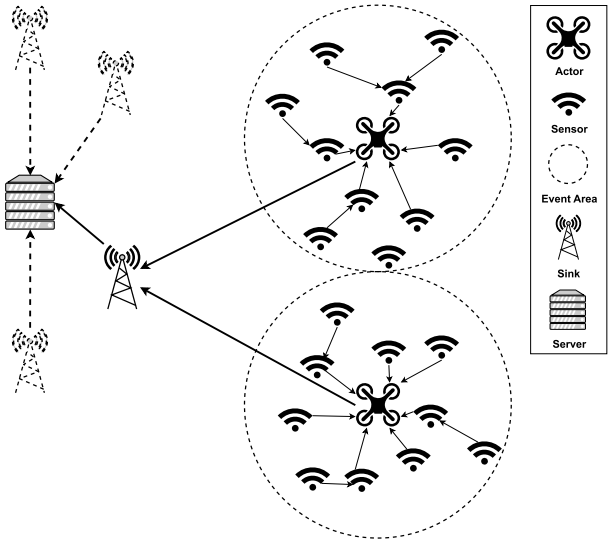


Fig. 2. Illustrative detail of sensors, mobile actors, sinks, and server in the architecture of a WSAN.

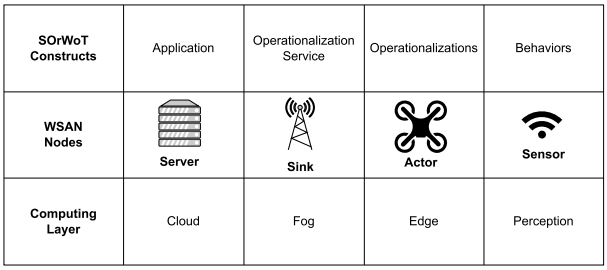


Fig. 3. Mapping of WSAN to SOrWoT constructs and computing layers.

reflects the hierarchical nature of both WSAN architectures and the SOrWoT framework. At the perception layer, sensors perform raw data collection and simple computations that correspond to the execution of low-level Behaviors in SOrWoT. Actors, positioned at the edge layer, handle more advanced processing and decision-making tasks, aligning with Operationalizations that coordinate multiple Behaviors. Sinks, operating at the fog layer, aggregate and preprocess data from multiple actors, and are mapped to Operationalization Services that manage alternative execution flows. Finally, servers at the cloud layer provide centralized analytics and long-term storage, corresponding to the Application construct in SOrWoT, which oversees system-wide Functionality. This layered mapping illustrates how SOrWoT constructs can be deployed across a WSAN infrastructure to support modular, adaptive, and distributed execution.

IV. WSAN OPTIMIZATION ENVIRONMENT

To exercise the use of different Operationalization types in a WSAN context and analyze their effects, a scenario was designed including a sink node, two actor nodes, and a set

of associated sensors per actor (see Figure 2). The following terms must be defined in this context:

- **Sensor**: *Associated Sensors* are assumed to be currently connected to an actor, while *Orphaned Sensors* refers to sensors that have lost communication to their previously associated actor. *Adopted Sensors* are previously orphaned sensors whose connection has been restored and are now associated to an actor;
- **Actor**: Mobile drones that select and process sensors' data, based on the currently selected Operationalization;
- **Coordinates**: Physical location of each sensing and actor entity, which is used to track the position of mobile actors and calculate their distance to each sensing device;
- **Sensor Noise**: Also referred to as read noise, represents the natural inaccuracies or variations introduced, at the sensor level, during the process of reading data;
- **Sensor Latency**: Time it takes for a communication of data from the sensor to reach the actor, and which is subject to natural fluctuations;
- **Ground Truth**: Actual sensor read value in an ideal scenario free from sensor noise. Used only during DQL agent training;
- **Operationalization Processed Reading**: The resulting sensor read value after the Operationalization has processed the sensors' data, in the current time step;
- **Operationalization Processed Latency**: The resulting delay considered after the Operationalization has selected and processed the sensors' data, in the current time step;
- **Read Error**: Quantifies the difference between the *processed reading* and *ground truth* temperature values.

Data gathered by sensors, i.e. read data, have variable accuracy due to sensor noise, which may lead to reading precision errors. Similarly, sensors have variable latency, i.e. the time they take to communicate the gathered data to the actor, and which, in our simulated environment, increases with the physical distance between the actor and the sensor.

Considering communication between the actor and its sensors using a query-driven model [44], where queries are launched across the network to retrieve the data, the actor can select, for each iteration of communication with its sensors, which type of Operationalization is executed, i.e. active, which basically determines what sensors' to use data from, and how to process that data. The type of Operationalization selected by the actor can provide advantages or disadvantages depending on the current environment conditions, namely the sensors' read noise level, and the sensors' latency fluctuations.

By simulating this environment, a DQL agent can be trained to minimize both read errors and latency. Also, when an actor node fails and its sensors become orphan, a recovery mechanism can be activated by the sink node to trigger a surviving close-by actor into action. However, as this actor repositions itself closer to the orphaned sensors to re-establish a connection with them, it is moving away from its original set of sensors. This means that, all sensors being considered, the average distance between the actor and the sensors is increased. During the failure event, and consequential distance increase between the actor and its sensors, the DQL must adapt and relearn the most advantageous Operationalization to use.

V. Markov Decision Process Model

An MDP is a mathematical framework for decision-making in stochastic environments, when an agent interacts with an environment to maximize long-term rewards, and is widely used for reinforcement learning. The MDP model is defined by the tuple $\langle S, A, P, R, \gamma \rangle$ where:

- S (State Space): Set of possible states that the agent can be in;
- A (Action Space): Set of possible actions that the agent can take;
- P (Transition Probability): Probability of reaching state s' after taking action a in state s; In deterministic environments, then $P(s' \mid s, a) = 1$;
- R (Reward Function): Immediate reward received after transitioning from state s to state s' due to action a;
- γ (Discount Factor): Determines the importance of future rewards. If $\gamma = 0$, agent only considers immediate rewards, and if $\gamma \approx 1$ it values longer-term rewards.

At each time step t, the agent:

1) Observes the current state s_t.
2) Selects an action a_t.
3) Observes environment response:
   - Receives a reward $R(s_t, a_t, s_{t+1})$.
   - Transitions to new state, s_{t+1}.
4) The process repeats until a terminal state or a fixed number of episodes is reached.

The goal of an MDP is to find an optimal policy $\pi^*(a \mid s)$ that maximizes the expected cumulative reward over time. The optimal policy is referred to as the *Bellman optimality equation* (see Equation 1).

$$V^{\pi*}(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi*}(s') \right) \quad (1)$$

where $V^{\pi*}(s)$ is the optimal value function for state s following policy π. The first term $R(s,a)$ is the immediate reward, while the second is the γ-discounted sum of future rewards assuming an optimal policy is followed by the agent.

In order to optimize sensor read error and latency, the following MDP model is defined and used to train a DQL agent.

A. State Space

Each state represents a unique configuration of the system. It is stored in the format of a state vector which encapsulates all the relevant information required to make decisions. This model stores the following information vectors, concatenated into a single one, with their length specified within parenthesis.

- Latency per sensor (θ)
- Reading noise per sensor (θ)
- Average actor-to-sensors distance (ρ)
- Latency per adopted sensor (ω)
- Reading noise per adopted sensor (ω)

where θ is the number of associated sensors, ρ is size one (the vector element for storing the distance value), and ω is the number of adopted sensors.

The adopted sensors fields exist in the state vector, as a placeholder for allowing an actor to be used for service restoration in case a nearby actor fails. Otherwise, they remain unused. These fields must exist in the state vector at the start of the simulation so that the DQN structure does not have to be recreated when the need for sensor adoption arises, i.e. in the event of actor failure, no more neurons need to be added to the neural network.

Each sensor must have its own slot in the vector for storing latency and reading noise values, since each sensor may be processed differently, depending on the active Operationalization type at a given time.

B. Action Space

The agent, which is represented by the actor, selects which Operationalization is currently active at each time step, with the intent of maximizing a reward. There are three possible choices, namely SER, POR, or PAN (see Sections I-A.1, I-B). Depending on the type of Operationalization the actor selects, sensor readings and latency are handled differently.

- Sensors' Reading Processing:
  - SER: Sensors' readings are processed one-by-one sequentially;
  - POR: Readings are processed in parallel. As soon as the *first* reading is available, it is used immediately without waiting for other sensors' data;
  - PAN: *All* readings are combined into a single result, averaged from all sensors' readings.
- Sensors' Latency Processing:
  - SER: The resulting latency is the *sum* of the latency of all sensors since they are processed sequentially;
  - POR: The latency is the *minimum* of all sensors, since only the fastest to respond is considered;
  - PAN: The *maximum* latency among sensors is considered, since they run in parallel and all must complete their task.

C. Reward Function

For each action taken in a given state, the agent observes the environment, and receives immediate feedback in the form of a reward, to guide the maximization of cumulative reward over time. The reward for this problem has two components, the read error and the latency (see Equation 2).

$$r = \alpha \cdot r^{\mathrm{E}} + (1 - \alpha) \cdot r^{\mathrm{L}} \tag{2}$$

where α is the weighting factor (see Table V). The read error reward, r^{E}, and latency reward, r^{L}, are calculated as per Equations 3 and 4 respectively.

$$r^{\mathrm{E}} = -|R^{\mathrm{OT}} - G| \tag{3}$$

$$r^{\mathrm{L}} = -\ln(L^{\mathrm{OT}}) \tag{4}$$

where R^{OT} is the reading value that results from the processing of sensor readings by the currently selected Operationalization type. The G is the ground truth value, that we know to be the correct measurement, and is used to compare against the processed reading and compute the absolute read error. L^{OT} is the Operationalization processed latency. The processed reading and processed latency consider one or more sensors' information depending on the current Operationalization type.

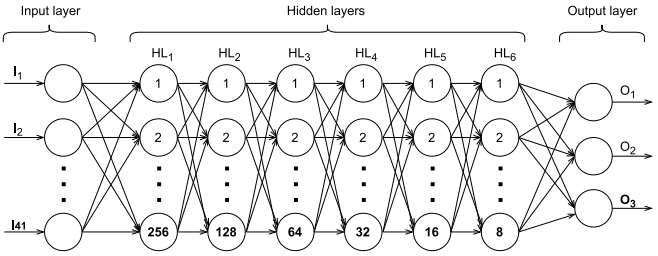


Fig. 4. DQN structure used to train the reinforcement learning agent.

Both read error and latency rewards carry a negative sign since we want to minimize these metrics. The read error is calculated in absolute terms in order to measure how far the reading is from the correct, ground truth value, regardless of whether it is above or below it. The logarithmic transformation in Equation 4 is used to calculate the latency reward in order to compress the range of large values, such as the processed latency from a SER Operationalization, and expand the range of smaller values such as the resulting latencies from POR and PAN Operationalizations. This makes small values more discernible for the agent during training.

VI. Deep Q-Network Model

DQL is a model-free, off-policy reinforcement learning algorithm that uses a deep neural network to approximate the Q-function. Model-free means the algorithm learns from interactions with the environment without previous knowledge of its dynamics such as state transition probabilities. Off-policy means the algorithm learns the optimal policy independently of the policy it is following, i.e. it learns from previous experiences stored in a replay buffer instead of directly from the actions the agent is taking [45]. The Q-function is a policy evaluation function that estimates the expected return of taking an action in a given state. The goal of DQL is to learn the optimal policy by maximizing the expected return over time [46].

Based on the defined MDP model, a DQL agent is developed to use the DQN algorithm, in order to optimize Operationalization selection in the environment. By using neural networks, DQN approximates the Q-value function, to evaluate the expected future rewards of state-action pairs.

This way, the agent uses DQN to iteratively refine its policy by exploring actions, observing rewards and updating its knowledge base, which are the weights and biases in the neural network. This process is what enables a reinforcement learning agent to learn optimal strategies.

The structure of the DQN used to solve this optimization problem is shown in Figure 4. In this network, the input layer has as many neurons as the size of the state vector. In total there are six hidden layers, where the first layer holds 256 neurons, being that number halved for each consecutive layer toward the output layer. Finally, the output layer contains

TABLE II
LIBRARIES AND TOOLS USED DURING DEVELOPMENT

Library/Tool	Version	Description
Keras	2.14.0	Neural networks in TensorFlow 2.
Matplotlib	3.9.4	Plotting library.
NumPy	1.23.5	Scientific computing library.
Python	3.9.21	Programming language.
VS Code	1.98.0	Integrated development environment.

TABLE III
WSAN SIMULATED ENVIRONMENT PARAMETERS

Parameter	Value	Description
Actor-to-Actor Distance	100	Measured in meters, is the distance between the two simulated actor nodes. Used to set actor 1 and 2's initial coordinates to [0,100], respectively, along the x-axis. This value makes it so that during normal operation, each actor is interacting with sensors up to a range that is half of the range configured in the communications protocol.
Communications Protocol Maximum Range	100	Measured in meters, defines the maximum distance that can exist between an actor and a sensor, when the actor radio signal is set to maximum power.
Communications Protocol Medium Range	50	Measured in meters, defines the maximum distance that can exist between an actor and a sensor, when the actor radio signal is set to medium power.
Failure Event Time	Episodes/2	Schedules the actor 2 to fail halfway through the simulation. This allows the division of the simulation into three stages, namely before failure, during failure, and after failure.
Ground Truth	25	Reference value in degrees Celsius, °C, represents the actual temperature reading and is used in the read error calculation. This value is only used during the agent's training, since a trained agent does not need the ground truth value to predict actions.
Noise Deviation	±2	Measured in degrees Celsius, °C, is the value of one standard deviation σ from the μ mean (ground truth value), in a normal Gaussian distribution of temperature readings. The noise is used to simulate real-world conditions where sensors are not perfectly accurate.
Sensors per Actor	10	Number of sensors associated with each actor at simulation start, and which the Operationalization can communicate with at each time step.

three neurons that correspond to the number of possible actions.

VII. ADAPTABLE OPERATIONALIZATION SIMULATION

In order to provide an environment for training the DQL agent, a custom simulator was developed using the Python programming language. A development using standard Python libraries was preferred over traditional established simulators, in order to provide the necessary flexibility in controlling experimental variables, and to include only the variables and functionalities relevant to our study, avoiding unnecessary complexity. Table II lists the most relevant tools and libraries used for the simulator development.

TABLE IV
DQL HYPERPARAMETERS

Param.	Value	Description	Justification
Batch Size	32	Defines the number of step memory records to use when training the DQN after each episode, using experience replay.	Chosen to balance computational efficiency with stable gradient updates: smaller batches risk noisy training, while larger ones demand excessive memory.
Discount Factor	0.5	Determines the importance of future rewards relative to immediate rewards.	Set to emphasize immediate rewards, since Operationalization choice in a step does not constrain future choices, making long-term discounting less critical.
Number of Episodes	4000	Controls overall training duration, and is the total number of full training sequences (episodes), each with a number of steps.	Selected to provide sufficient training coverage for convergence while keeping computational cost manageable.
Epsilon Decay	6.6×10^{-4}	Controls the rate at which exploration decreases from its maximum ($\epsilon = 1$) to zero across training episodes.	Decay value is calculated as $1/(\frac{Nr.Episodes}{2} \cdot \frac{3}{4})$ so that exploration decreases to zero within 75% of the episodes in each stage, leaving 25% for exploitation both before and after failure.
Learning Rate	1.0×10^{-4}	Defines the step size used to update the neural network's weights and biases.	Set to a small value to ensure incremental updates, preventing overshooting and maintaining stability.
Steps per Episode	10	The number of agent–environment interactions (action, state, reward) until the episode ends.	Fixed at 10 to keep episodes short and computationally efficient, while still allowing meaningful sequences of decisions.

A. Simulation Data and Logic

The simulation setup combines both the data employed during training and the algorithmic logic used in the optimization. Since the training process is based on reinforcement learning, no labelled dataset is used—as would be the case for supervised learning. Instead, the agent uses data dynamically generated through its interaction with the environment. In particular, sensor reading noise and sensor latency are incorporated to emulate realistic network conditions (see Section VII-B). These generated data are explicitly used in the agent's reward function (see Equations 3 and 4), and in the training process (see step 19 in Algorithm 1).

1) Environment Parameters: The environment is instantiated using the parameters listed in Table III. These parameters are used to generate the environment's elements and their positioning, failure events, and to calculate metrics. Together, they provide the structural and stochastic elements that shape the agent's experience during training.

2) DQL Agent Parameters: The agent's learning process relies on hyperparameters tuned through extensive experimentation within generally accepted ranges [47]. The final configuration was selected to promote stable convergence and effective learning performance. Training was conducted over 4000 episodes, each consisting of 10 steps, with the hyperparameter values summarized in Table IV.

For completeness, a compact overview of the training setup—including hardware, number of episodes, Epsilon

Algorithm 1 Simulated Training of DQL Agent in Actor 1

```
Setup environment parameters
Setup DQL training limits and hyperparameters
Initialize wireless communications protocol (medium and maximum signal range)
Initialize actor 1 and actor 2 instances
Generate cluster of sensors randomly positioned, for both actor 1 and actor 2, within their medium signal range
Compute average distance of actor 1 to its sensors
Initialize the DQL agent
for episodes do
    Reset the environment
    for steps do
        if actor 2 failed then
            Sink triggers recovery actions on actor 1
            Actor 1 computes new center considering all sensors including orphaned ones
            Actor 1 moves to new coordinates
            Actor 1 adopts orphaned sensors
            Reset DQL agent's Epsilon hyperparameter
            Reset actor 2 failed flag
        end if
        Get sensor noise and latency from state vector
        Select action (Operationalization type to use)
        Compute resulting read error per sensor
        Compute resulting latency per sensor
        Compute total reward
    end for
    Train DQN to better predict Q-values
end for
```

schedule, reset condition, learning rate, and reward coefficients—is provided in Table V.

3) Simulation Logic: The overall simulation flow for training the DQL agent, executed in actor 1, for optimizing the WSAN environment is described in Algorithm 1. The agent is trained using mini-batch gradient descent and experience replay, ensuring stable convergence and efficient use of past experience. The interaction between the generated data, the environment parameters, and the DQL agent's parameters, forms the basis of the simulation process.

B. Simulation of Network Conditions

In order to enhance the realism of the simulated scenario, network conditions are included to introduce elements of unpredictability. Specifically, the concepts of sensor noise and latency fluctuations are used. By considering noise in sensor readings, we account for the natural accuracies in sensor data. Similarly, the time it takes for the sensor data to reach the actor is subjected to fluctuations. Together, these conditions contribute to a more comprehensive representation of network behavior.

1) Sensor Reading Noise: A noise component is added to each sensor reading data. This value is randomly generated according to a normal Gaussian distribution, where the mean is a predefined ground truth value and a standard deviation of ± 2 °C is used, meaning $\approx$ 68% of readings fall into this interval (see Table III). Sensor noise affects the read data, and consequently the read error magnitude (see R^{OT} in Equation 3).

TABLE V
TRAINING SETUP

Parameter	Value/Description
Hardware	Intel Core i7-6700, 15.5 GiB RAM; NVIDIA GeForce RTX 2080 Ti, 10.5 GiB VRAM
Number of Episodes	4000
Epsilon Schedule	Linear decay from 1.0 to 0.0 over 75% of episodes (per stage, before and after failure); final 25% used for exploitation.
Reset Condition	Environment is reset after each episode of 10 steps; Epsilon is reset after actor failure.
Learning Rate	1.0×10^{-4}
Reward Coefficients	Apply $\alpha = 0.5$

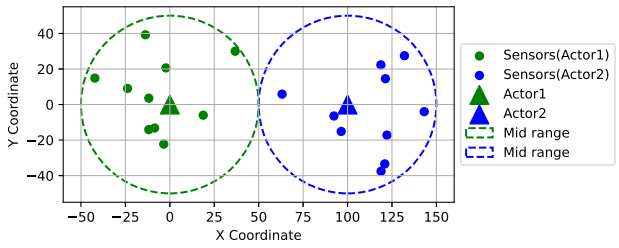


Fig. 5. Actors and sensors coordinates in stage 1 (normal operation).

2) Single Sensor Latency: Each communication of data from a sensor to the actor has an associated latency. Sensor latency, $L_s \in [0, 1]$, in this simulation is assumed to be partially affected by the distance between the sensor and actor nodes, and is calculated according to Equation 5.

$$L_s = 0.8 * \tau_s^{\mathrm{B}} + 0.2 * \tau_s^{\mathrm{R}} \tag{5}$$

$$\tau_s^{\mathrm{B}} = d_s^{\mathrm{CUR}} / d^{\mathrm{MAX}} \tag{6}$$

where $\tau_s^{\mathrm{B}} \in [0, 1]$ is the base delay for sensor s, which we define to depend on the current distance between sensor and actor, d_s^{CUR}, and is computed as per Equation 6. d^{MAX} is the maximum signal distance allowed in the communication protocol in use. τ_s^{R} is a random delay which has less influence on the sensor latency than the base delay and does not depend on the distance. Its weight of 0.2 is justified by the influence of natural environment factors on data transmission [48].

C. Simulation of Actor Failure Event

The simulation flows through three different stages to provide the circumstances for the agent to learn how to optimize the relevant metrics, but also to adapt to a change of network conditions after an actor failure.

The simulation starts with the first stage, where actors 1 and 2 in the WSAN are working under normal operation conditions, each receiving and processing data from their associated sensors. In this stage, actor nodes are at medium capacity of their signal transmission power (see Figure 5).

In the second stage, actor 2 stops operating due to a fault. As a response, a sink node tasks actor 1 with performing recovery steps. To complete this assignment, actor 1 needs to consider the orphaned sensors' coordinates, and calculate the best place

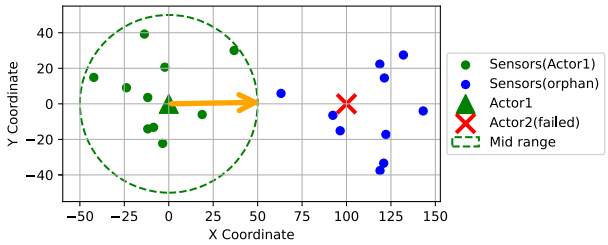


Fig. 6. Actors and sensors coordinates in stage 2 (actor failure and recovery).

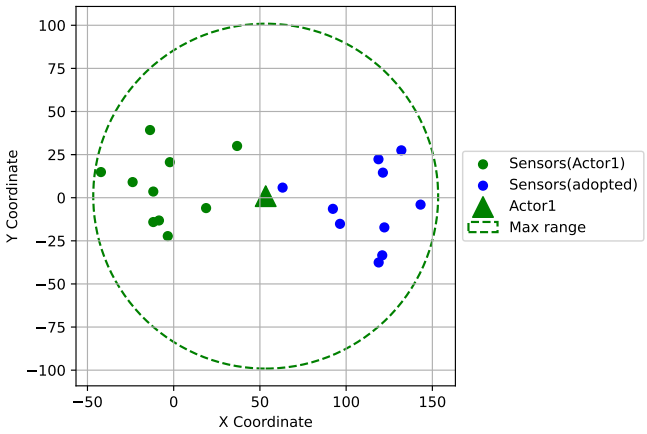


Fig. 7. Actors and sensors coordinates in stage 3 (recovered operation).

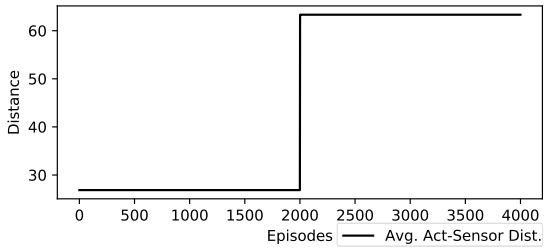


Fig. 8. Average distance between actor and its sensors over training episodes.

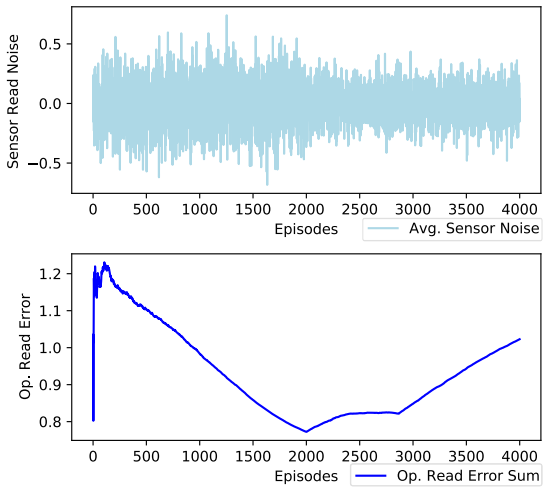


Fig. 9. Sensor read noise (top) and Operationalization resulting read error (bottom) over training episodes.

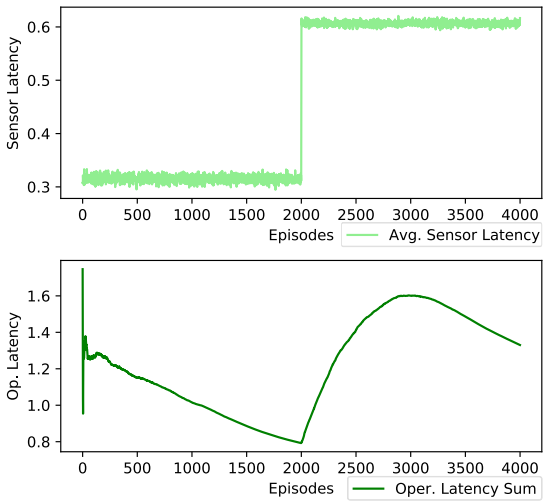


Fig. 10. Sensor latency (top) and Operationalization resulting latency (bottom) over training episodes.

to move to, in order to minimize the average distance between it and all sensors, including both its original ones and the ones left by the failed actor 2 (see Figure 6).

In stage 3, the last stage, normal operation is recovered by having actor 1 move to the new center coordinates relative to all sensors, increase its signal transmission power to cover all of them in its event area, and adopt the orphaned sensors to start receiving and processing their data (see Figure 7).

VIII. Analysis of Results

During the DQL agent's training for optimizing data accuracy and latency across the aforementioned stages, multiple data points were collected which allowed a better understanding of the environment's changes and their effects on the agent's behavior.

During the third stage of the simulation, in response to actor 2's failure, and triggered by the sink node's network recovery request, actor 1 computed and moved to new coordinates in order to minimize its distance to all sensors, including its original ones, and the ones left orphaned by actor 2. As actor 1 adopted the additional sensors, at episode 2000 of the training, the average distance to its associated sensors was recalculated and consequently increased. Figure 8 shows how this distance increased from just under 30 to over 60, more than doubling, which suggests that communication latency between the actor and its sensors is likely to increase accordingly.

The simulated environment's sensor reading noise, in Figure 9 (top), and sensor latency in Figure 10 (top) were generated including natural variations to make the environment more realistic. The DQN agent needs to filter out such stochasticity in its search for the actions that ultimately improve rewards. Figure 9 (top) shows how the step-averaged reading noise varies over episodes according to a Gaussian distribution with the ground truth value as mean (see Table III), and a standard deviation of ± 2 °C. Sensor latency, as depicted in Figure 10 (top), is susceptible to environmental interference causing its small scale variations, and sustained its greatest shift after actor 1 repositioned itself in episode 2000, resulting in a change in actor-sensor distance as per Figure 8. This event caused the step-averaged latency per episode to almost double from around 0.32 to 0.61 (see Figure 10 (top)).

Figures 9 (bottom) and 10 (bottom) show the resulting read error and resulting latency, respectively. These values, plotted as their individual sums for the steps in each episode, are

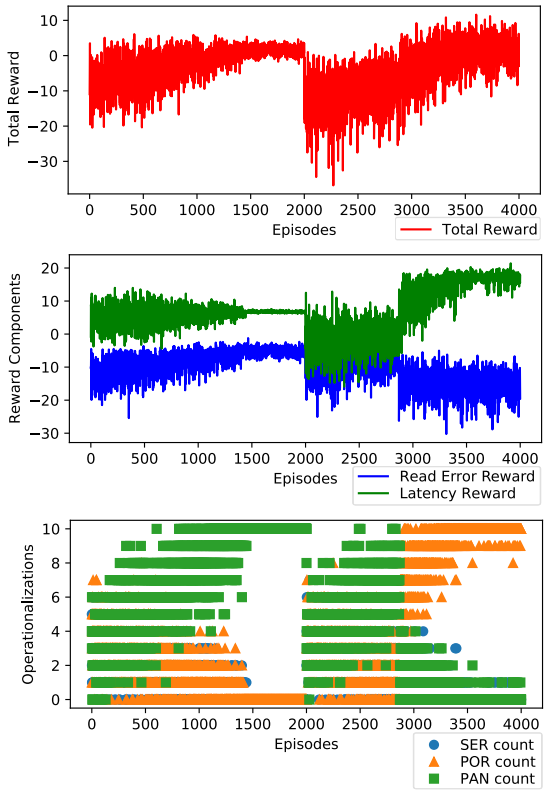


Fig. 11. Total agent reward (top), read error and latency reward components (middle), and selected Operationalizations (bottom) over training episodes.

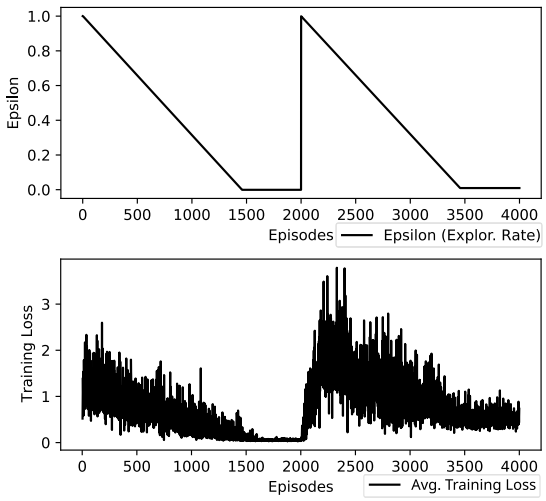


Fig. 12. Epsilon agent exploration rate (top), DQN model loss (bottom) over training episodes.

computed as the sensors' output read error and latency per step, which result after the currently active Operationalization in actor 1 has processed the data received from its sensors. In these plots, it can be observed how the agent experiments with policies for the first couple of hundred episodes, searching for a way to decrease the resulting read error and latency. It eventually finds a way to start minimizing both at around episode 200, and keeps improving both metrics until the actor 2's failure, which leads to the previously mentioned distance increase. The successful minimization of both sensor read error and latency is achieved through the convergence of the DQN policy, which selected an Operationalization type that balances accuracy from multiple sensor readings with moderate data processing latency. For this purpose, the agent converged on PAN (see Figure 11 (bottom)).

Following the failure event at episode 2000, the agent's optimization performance deteriorated for both metrics, although more significantly for latency (see Figure 10 (bottom)), most likely due to the sharp increase in distance (see Figure 8) interfering with the latency result. After about 900 episodes of an increasing read error and latency, around episode 2900 the agent learned that it was more advantageous to start focusing on improving the resulting latency *even* at the expense of the read error result. That is the reason why from that point on, latency started to decrease, while read error initiated a sharper increase. The read error increase is visible from episode 2900 in Figure 9 (bottom), while the latency decrease—starting from the same episode—can be seen in Figure 10 (bottom). This episode marks the DQN policy shift from PAN to POR, prioritizing speed over accuracy (see Figure 11 (bottom)).

Figure 11 depicts the total reward (top), the reward components (middle), and the Operationalizations chosen by the agent (bottom) over the training episodes.

The total reward per episode is calculated as the sum of the rewards for each step in that episode. That reward can be seen to improve gradually in stage 1 of the simulation (normal operation) up to episode 2000, having both read error and latency reward as its summed components (see Figure 11 (middle)). The agent managed to achieve this improvement, before any failure occurred, by mainly selecting to use the PAN Operationalization (green squares in Figure 11 (bottom)) to select and process sensors' data, prioritizing accuracy above latency. Figure 11 (bottom), shows the number of times each Operationalization was used during the episode, i.e. in the 10 steps. If a marker is drawn close to the top, for a count of 10 Operationalizations, it means the agent chose to use that Operationalization type in every step of that episode. This figure clearly shows the two different stages of learning, the first, before the failure occurred, converging to PAN, and the second stage, after actor 2 failed, starting from episode 2000, which eventually converged to select the POR Operationalization when under greater latency constraints.

It is worth noting that frequent switching between Operationalization modes occurs during the training phase. After convergence, such changes may still arise in the event of actor node failure and relocation, which can alter sensor-actor distances and may require further considerations such as sensors energy redistribution.

After the failure and recovery procedure of stage 2, in stage 3 after episode 2000, the total reward experienced a significant drop (see Figure 11 (top)), with the latency reward component having the biggest impact (see Figure 11 (middle)), mostly caused by the distance increase. To a lesser extent, the read noise reward component also worsened at the start of stage 3, although this worsening can be explained as an effect of the reset triggered on the exploration rate (see *Epsilon Schedule* in Table V), performed in order to allow the agent to

TABLE VI
OPERATIONALIZATIONS SELECTED BY ACTOR 1 BEFORE AND AFTER THE FAILURE OF COMMUNICATION IN ACTOR 2'S SENSORS UNDER DIFFERENT FAILURE RATES

Scenario	Sensor-Actor2 Failure Rate	Selected Oper. Pre-Failure	Selected Oper. Post-Failure
A	0	PAN	PAN
B	0.25	PAN	POR
C	0.50	PAN	POR
D	0.75	PAN	POR
E	1	PAN	POR

better adapt to the new environment conditions (see Figure 12 (top)).

Starting at around episode 2900, and in line with the resulting noise and latency shown in Figures 9 and 10, the agent managed to improve its total reward, by choosing to use the POR Operationalization to process sensor data in most of its steps. This is visible by the growing Operationalization convergence into orange triangles in Figure 11 (bottom). This policy, which is different from the one the agent used for the first stage of the simulation, considered latency reward to be the most impactful component for the total reward, given the new environment conditions.

Although the agent consistently overlooked the SER Operationalization type due to its lower efficiency in the simulated environment (see Figure 11 (bottom)), it may still be viable under specific conditions. In a query-driven communication model operating within constrained networks, sequential execution reduces simultaneous sensor queries, thereby mitigating message bursts to which such networks are highly vulnerable. Moreover, SER's ordered execution can be advantageous for temporal profiling or historical trend detection, where preserving the sequence of sensor readings provides interpretability and cumulative insights. Thus, while SER is generally suboptimal for latency or accuracy-sensitive WSANs, alternative reward functions or environments emphasizing communication efficiency or temporal continuity could favor its selection.

Figure 12 shows the evolution of the agent's exploration rate (top) and training loss (bottom). During stage 1 (before episode 2000) and stage 3 (after episode 2000), the exploration rate started at its maximum value, was decayed linearly, and eventually reached zero at approximately three-quarters of the stage's duration. The sharp increase of the exploration rate at episode 2000 is caused by the reset of the Epsilon hyperparameter, triggered by the detected change in average actor-sensors distance, and used to allow the agent to explore an alternative policy.

Training loss (Figure 12 (bottom)) decreased both during stage 1 and 3, representing the updating of weights and biases in the DQN. Throughout training, the loss value did not level off at zero, nor it dropped too quickly, meaning that the neural network's capacity was adequate for preventing overfitting to the training data. On the other hand, the gradual decrease in loss, instead of increase, indicates that underfitting was also avoided in this model.

A. Sensitivity to Partial Failures

To further analyze the robustness of the learned DQN policy, a set of five scenarios was conducted, simulating increasing levels of sensor failure in actor 2, ranging from 0 to 1, in 0.25 increments. By sensor failure we mean communication failures caused by unstable wireless connections or environmental interference. A higher failure rate means that more sensors lose connection to actor 2, implying that actor 1 must adopt more orphaned sensors, leading it to reposition closer to the geometric center of its extended sensor set. Table VI summarizes the Operationalization types selected by the agent before and after the failure event in each scenario. Results show that when the failure rate is low (0%), the agent consistently maintains the PAN Operationalization both before and after the failure. However, from approximately 25% sensor failure onward, the agent begins switching to the POR Operationalization after the failure. This transition suggests that as more sensors must be associated to a single actor, the agent prioritizes lower data collection latency over accuracy. These findings show that the learned policy adapts Operationalization strategies based on the severity of network degradation and associated latency-accuracy trade-offs.

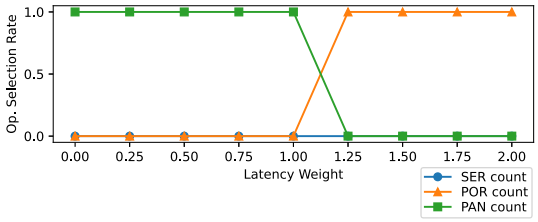


Fig. 13. A trained agent's Operationalization selection rate varies with latency weight, while reading error weight is kept fixed.

B. Sensitivity to Latency Weight

Figure 13 shows the results of an ablation experiment in which the agent was trained under normal conditions for several scenarios where the read error weight in the reward function was kept fixed (see Equation 2) and the latency weight was varied in 0.25 increments between 0 and 2. Results show that when latency has low influence, the agent favors PAN, which minimizes read error through sensor averaging. As latency weight increases, the policy shifts to POR between weight 1 and 1.25, indicating the point where latency becomes the dominant factor driving the agent's decision-making.

IX. CONCLUSION AND FUTURE WORK

In this study, we set out to design an automated solution for optimizing data accuracy and latency in a WSAN, by exploring the advantages of using different Operationalization types, made available by the SOrWoT framework. This solution needed to work under variable network conditions, and be tolerant to actor node failures. The main contributions of this work can be summarized as follows. First, we provide a conceptual mapping of SOrWoT constructs to WSAN elements, establishing a clear correspondence between abstract concepts and their practical counterparts. Building on this foundation, we introduce an MDP formulation that guides the selection of the most suitable Operationalization type. We then design a DQL agent capable of dynamically adapting to environmental

conditions such as noise, latency and actor failures. To support this, we develop a custom simulation environment together with a DQL training algorithm. Lastly, we present an in-depth analysis of the agent's optimization performance under variable conditions, demonstrating the robustness and applicability of the proposed approach.

During the agent's training, initially the most rewarding metric to optimize was the read error, to some extent due to latency being naturally lower for shorter distances between actor and sensors. However, after the actor failure, and consequent increase in distance between the surviving actor and all sensors, including the adopted ones, the scale tipped in favor of minimizing latency. So under the new conditions, latency became a greater issue, meaning more potentially rewarding for the agent, when compared to the read error which was not affected by the change in distance.

According to the agent's learned policies, throughout all training, the SER Operationalization was largely overlooked in favor of more rewarding options, such as PAN and POR. The rationale behind this is that before the actor failure (stage 1), PAN was mostly active, and able to minimize read error by averaging data readings from multiple sensors, since errors tend to cancel each other out in a normal distribution. On the other hand, SER and POR Operationalizations resulted in greater variability, meaning a less accurate estimate of the true mean, since only the last or first sensors were considered.

The failure event and the recovery steps (stage 2) triggered a reset of the agent's exploration rate, allowing it to reconsider its policy under the new conditions. After the surviving actor restored operation and adopted the orphaned sensors (stage 3), the average distance to its sensors was consequentially increased, and the agent adapted dynamically by learning that the most appropriate Operationalization was POR at that point.

As the impact of latency on the agent's reward became greater due to increased distance, using a POR Operationalization meant that speed was prioritized over data accuracy, with the actor considering only data from its fastest sensor. Other Operationalization types exhibited lower performance, namely with PAN processing taking as long as the slowest sensor, and an even slower SER, whose sequential nature resulted in a total latency equal to the sum of all individual sensor latencies.

Overall, while SER Operationalization was consistently suboptimal in the studied environment, it may still hold value in specific constrained network scenarios, where sequential execution reduces message bursts compared to parallel approaches, or in temporal profiling tasks that benefit from ordered sensor readings. Nevertheless, in latency or accuracy-sensitive environments parallel strategies remain superior.

The agent's adaptability was further confirmed through two sensitivity analyses. The study on sensitivity to partial failures showed that as the proportion of failed sensors increased, the policy transitioned from PAN to POR, reflecting the agent's ability to respond to network degradation. The ablation experiment on sensitivity to latency weight revealed a similar shift as the latency reward was progressively emphasized, identifying the trade-off point at which latency outweigh accuracy.

By integrating reinforcement learning with SOrWoT Operationalization constructs, this study showed that intelligent agents can autonomously optimize WSAN behavior under dynamic and failure-prone conditions. These findings contribute to the development of resilient, self-adaptive IoT systems.

Future work will extend this approach to deeper Operationalization architectings, multi-actor coordination, and explicit energy and routing considerations, enabling a more comprehensive evaluation of decision-making and network lifetime under realistic operational constraints.

REFERENCES

[1] A. K. Vishwakarma, S. Chaurasia, K. Kumar, Y. N. Singh, and R. Chaurasia, "Internet of Things technology, research, and challenges: A survey," *Multimedia Tools Appl.*, vol. 84, no. 11, pp. 8455–8490, May 2024.

[2] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, document RFC 7252, Jun. 2014.

[3] D. Bandyopadhyay and J. Sen, "Internet of Things: Applications and challenges in technology and standardization," *Wireless Pers. Commun.*, vol. 58, no. 1, pp. 49–69, May 2011.

[4] G. Singh and S. Sharma, "A comprehensive review on the Internet of Things in precision agriculture," *Multimedia Tools Appl.*, vol. 84, no. 17, pp. 18123–18198, Jul. 2024.

[5] E. M. B. M. Karunathilake, A. T. Le, S. Heo, Y. S. Chung, and S. Mansoor, "The path to smart farming: Innovations and opportunities in precision agriculture," *Agriculture*, vol. 13, no. 8, p. 1593, Aug. 2023.

[6] N. Kitpo and M. Inoue, "Early Rice disease detection and position mapping system using drone and IoT architecture," in *Proc. 12th South East Asian Tech. Univ. Consortium (SEATUC)*, vol. 1, Mar. 2018, pp. 1–5.

[7] O. Friha, M. A. Ferrag, L. Shu, L. Maglaras, and X. Wang, "Internet of Things for the future of smart agriculture: A comprehensive survey of emerging technologies," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 4, pp. 718–752, Apr. 2021.

[8] R. Guebsi, S. Mami, and K. Chokmani, "Drones in precision agriculture: A comprehensive review of applications, technologies, and challenges," *Drones*, vol. 8, no. 11, p. 686, Nov. 2024.

[9] R. R. Shamshiri et al., "Research and development in agricultural robotics: A perspective of digital farming," *Int. J. Agricult. Biol. Eng.*, vol. 11, no. 4, pp. 1–11, Jul. 2018.

[10] C. L. D. Abreu and J. P. V. Deventer, "The application of artificial intelligence (AI) and Internet of Things (IoT) in agriculture: A systematic literature review," in *Proc. Southern Afr. Conf. Artif. Intell. Res.*, Jan. 2022, pp. 32–46.

[11] X. Zhang, Y.-L. Wang, and H. Byun, "Divisive hierarchical clustering for energy saving and latency reduction in UAV-assisted WSANs," *EURASIP J. Wireless Commun. Netw.*, vol. 2025, no. 2, Jan. 2025.

[12] K. S. Mohamed, "IoT physical layer: Sensors, actuators, controllers and programming," in *The Era of Internet of Things*, 2019, pp. 21–47.

[13] R. Gomes and N. Correia, "On a self-orchestrated Web of things (SOrWoT)," *TechRxiv*, Dec. 2024, doi: 10.36227/techrxiv.173385990.01916909/v1. [Online]. Available: https://www.techrxiv.org/users/862115/articles/1246780-on-a-self-orchestrated-webof-things-sorwot

[14] (Dec. 2023). *W3C Web of Things (WoT) Thing Description 1.1*. [Online]. Available: https://www.w3.org/TR/wot-thing-description11/

[15] K. J. Naidu, K. V. Babu, C. R. C. Sai, P. Ganesh, T. G. Sai, and N. S. Naidu, "Precision agriculture monitoring system," *Biosci. Biotechnol. Res. Asia*, vol. 21, no. 2, pp. 1543–1551, Jun. 2024.

[16] A. Soussi, E. Zero, R. Sacile, D. Trinchero, and M. Fossa, "Smart sensors and smart data for precision agriculture: A review," *Sensors*, vol. 24, no. 8, p. 2647, Apr. 2024.

[17] U. Shafi, R. Mumtaz, J. García-Nieto, S. A. Hassan, S. A. R. Zaidi, and N. Iqbal, "Precision agriculture techniques and practices: From considerations to applications," *Sensors*, vol. 19, no. 17, p. 3796, Sep. 2019.

[18] M. S. Basingab et al., "AI-based decision support system optimizing wireless sensor networks for consumer electronics in e-commerce," *Appl. Sci.*, vol. 14, no. 12, p. 4960, Jun. 2024.

[19] V. Barrile, C. Maesano, and E. Genovese, "Optimization of crop yield in precision agriculture using WSNs, remote sensing, and atmospheric simulation models for real-time environmental monitoring," *J. Sensor Actuator Netw.*, vol. 14, no. 1, p. 14, Jan. 2025.
[20] S. Singh and R. Sharma, "Threshold-sensitive energy efficient routing for precision agriculture," *Peer-Peer Netw. Appl.*, vol. 18, no. 3, p. 155, May 2025.
[21] H. Jawad, R. Nordin, S. Gharghan, A. Jawad, and M. Ismail, "Energy-efficient wireless sensor networks for precision agriculture: A review," *Sensors*, vol. 17, no. 8, p. 1781, Aug. 2017.
[22] F. Mikolajczak, B. Schnor, and G. M. Huamán, "Sensor fault diagnosis for precision agriculture," *GI/ITG KuVS Fachgespräch Sensornetze*, p. 28, 2023. [Online]. Available: https://www.cs.unipotsdam.de/bs/research/docs/papers/2023/mhs23.pdf
[23] N. C. Luong et al., "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surv. Tut.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.
[24] C. Zieliński, "Specification of agent based robotic systems using hierarchical finite state automatons," in *Proc. The 20th Polish Control Conf. Adv., Contemp. Control*. Cham, Switzerland: Springer, 2020, pp. 465–476.
[25] M. Klotzbücher and H. Bruyninckx, "Coordinating robotic tasks and systems with rFSM statecharts," *J. Softw. Eng. Robot.*, vol. 3, no. 1, pp. 28–56, 2012.
[26] S. Sarwar, R. Sirhindi, L. Aslam, G. Mustafa, M. M. Yousaf, and S. W. U. Q. Jaffry, "Reinforcement learning based adaptive duty cycling in LR-WPANs," *IEEE Access*, vol. 8, pp. 161157–161174, 2020.
[27] Y. Liu, J. Yan, and X. Zhao, "Deep-reinforcement-learning-based optimal transmission policies for opportunistic UAV-aided wireless sensor network," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13823–13836, Aug. 2022.
[28] K. Toyoshima, T. Oda, M. Hirota, K. Katayama, and L. Barolli, "A DQN based mobile actor node control in WSAN: Simulation results of different distributions of events considering three-dimensional environment," in *Proc. 8th Int. Conf. Emerg. Internet, Data Web Technol. Adv. Internet Data Web Technol.* Cham, Switzerland: Springer, Jan. 2020, pp. 197–209.
[29] R. Piyare et al., "Integrating wireless sensor network into cloud services for real-time data collection," in *Proc. Int. Conf. ICT Converg. (ICTC)*, Oct. 2013, pp. 752–756.
[30] T. Dinh and Y. Kim, "An efficient sensor-cloud interactive model for on-demand latency requirement guarantee," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, May 2017, pp. 1–6.
[31] X. Liu, A. Liu, T. Qiu, B. Dai, T. Wang, and L. Yang, "Restoring connectivity of damaged sensor networks for long-term survival in hostile environments," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 1205–1215, Feb. 2020.
[32] B.-M. Cho, S. Kim, K.-D. Kim, and K.-J. Park, "A controller switching mechanism for resilient wireless sensor–actuator networks," *Appl. Sci.*, vol. 12, no. 4, p. 1841, Feb. 2022.
[33] K. Mukherjee, S. Gupta, A. Ray, and T. A. Wettergren, "Statistical-mechanics-inspired optimization of sensor field configuration for detection of mobile targets," *IEEE Trans. Syst., Man, Cybern., B (Cybern.)*, vol. 41, no. 3, pp. 783–791, Jun. 2011.
[34] O. Kaiwartya et al., "Virtualization in wireless sensor networks: Fault tolerant embedding for Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 571–580, Apr. 2018.
[35] S. C. Kadu, M. Bhushan, and R. Gudi, "Optimal sensor network design for multirate systems," *J. Process Control*, vol. 18, no. 6, pp. 594–609, Jul. 2008.
[36] F. Marcelloni and M. Vecchio, "Enabling energy-efficient and lossy-aware data compression in wireless sensor networks by multi-objective evolutionary optimization," *Inf. Sci.*, vol. 180, no. 10, pp. 1924–1941, May 2010.
[37] M. Ayar, A. Isazadeh, F. S. Gharehchopogh, and M. Seyedi, "NSICA: Multi-objective imperialist competitive algorithm for feature selection in arrhythmia diagnosis," *Comput. Biol. Med.*, vol. 161, Jul. 2023, Art. no. 107025.
[38] N. Khodadadi et al., "Multi-objective generalized normal distribution optimization: A novel algorithm for multi-objective problems," *Cluster Comput.*, vol. 27, no. 8, pp. 10589–10631, May 2024.
[39] N. Khodadadi, F. Soleimanian Gharehchopogh, and S. Mirjalili, "MOAVOA: A new multi-objective artificial vultures optimization algorithm," *Neural Comput. Appl.*, vol. 34, no. 23, pp. 20791–20829, Aug. 2022.
[40] A. Heydariyan, F. S. Gharehchopogh, and M. R. E. Dishabi, "A hybrid multi-objective algorithm based on slime mould algorithm and sine cosine algorithm for overlapping community detection in social networks," *Cluster Comput.*, vol. 27, no. 10, pp. 13897–13917, Jul. 2024.
[41] N. Khodadadi, F. S. Gharehchopogh, B. Abdollahzadeh, and S. Mirjalili, "AMHS: Archive-based multi-objective harmony search algorithm," in *Proc. 7th Int. Conf. Harmony Search*, Sep. 2022, pp. 259–269.
[42] F. S. Gharehchopogh, B. Abdollahzadeh, S. Barshandeh, and B. Arasteh, "A multi-objective mutation-based dynamic Harris Hawks optimization for botnet detection in IoT," *Internet Things*, vol. 24, Dec. 2023, Art. no. 100952.
[43] J. Bhola and S. Soni, "A study on research issues and challenges in WSAN," in *Proc. Int. Conf. Wireless Commun., Signal Process. Netw. (WiSPNET)*, Mar. 2016, pp. 1667–1671.
[44] X. Ma et al., "A survey on data storage and information discovery in the WSANs-based edge computing systems," *Sensors*, vol. 18, no. 2, p. 546, Feb. 2018.
[45] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
[46] J. Clifton and E. B. Laber, "Q-learning: Theory and applications," *Annu. Rev. Statist. Its Appl.*, vol. 7, no. 1, pp. 279–301, Mar. 2020.
[47] Y. Lu, "Comparison and hyperparameter analysis of four reinforcement learning algorithms in the lunar lander environment," in *Proc. Int. Conf. Artif. Intell.*, Jun. 2024, pp. 257–266.
[48] C. S. Tezcan and K. S. Gundogdu, "The effect of meteorological parameters on wireless data transmission," *J. Agricult. Nature*, vol. 25, pp. 1127–1133, Oct. 2022.

Ruben Gomes received the B.Sc. and M.Sc. degrees in computer science and systems engineering from the University of Algarve, Faro, Portugal, in 2006 and 2013, respectively. He is currently pursuing the Ph.D. degree in computer science (IoT networks optimization). He is an Assistant Lecturer with the Science and Technology Faculty, University of Algarve. He is a Research Fellow with the Networks and Systems Group, Center for Electronics, Optoelectronics and Telecommunications, supported by Portuguese Foundation for Science and Technology. His research interests include the IoT, machine learning, and network optimization.

Noélia Correia received the B.Sc. and M.Sc. degrees in computer science from the University of Algarve, Faro, Portugal, in 1995 and 1998, respectively, and the Ph.D. degree in optical networks (computer science) from the University of Algarve in 2005, in collaboration with the University College London, U.K.

She is a Lecturer with the Science and Technology Faculty, University of Algarve. She is a Founding Member of the Center for Electronics, Optoelectronics and Telecommunications, University of Algarve, a research center supported by Portuguese Foundation for Science and Technology. She is also the Networks and Systems Group Coordinator. Her research interests include the application of optimization techniques to several network design problems, in the optical, wireless, sensor and IoT networks fields, and development of algorithms.