SUP'COM
Higher School of Communication of Tunis

المدرسة العليا للمواصلات بتونس
ÉCOLE SUPÉRIEURE DES COMMUNICATIONS DE TUNIS

Telecommunications engineering cycle

Major:

Mobility University of Padova

Graduation Project Report

Topic:

Intelligent Scheduling in Wake-up Radio Systems

Realized by:

Fares Khelifi

Supervisors:

Dr. Federico Chiariotti
Dr. Ons Ben Rhouma

Work proposed and realized in collaboration with:

Università degli Studi di Padova

Academic year: 2022 - 2023

Abstract

The management of an intelligent scheduling system for wake-up radio within Internet of Things (IoT) networks is a paramount task, essential to the sustainable operation of these systems. Wake-up radio technology, as a critical part of IoT, provides unique scheduling opportunities to optimize energy consumption and extend sensors batteries life. The ability to construct intelligent scheduling requests, by integrating knowledge on the system state, statistical data, and the sensor battery states and capabilities, is central to achieving these goals.
This energy efficiency is not only instrumental in extending the longevity of the IoT devices, but also plays a crucial role in enhancing the overall system performance. By reducing the energy demand, the network can function more effectively for longer periods, thereby improving the reliability of the IoT system.
The advancement in reinforcement learning, and specifically Deep Q-Network (DQN) algorithms, provides an efficient tool for implementing this intelligent scheduling. Through training an agent using these techniques, we can improve the agent's capability to differentiate between events, so we can optimize the scheduling process. This refined differentiation is a significant advantage as it further conserves energy and enhances system reliability.
The proposed scheduling framework and algorithms are thoroughly evaluated through a series of rigorous simulations. These simulations help validate the effectiveness of the approach and provide insights into potential areas of further improvement.
The study underlines the promising potential of this intelligent scheduling framework for wake-up radio systems. It marks a significant contribution towards the development of more sustainable and resilient IoT ecosystems. As the world moves towards increased digitization, the benefits of such optimized IoT systems cannot be overstated, with implications for a wide range of applications in sectors such as smart homes, healthcare, transportation, and more.

Acknowledgement

I would like to express my profound gratitude to my home institution, SUP'COM, for providing me the opportunity to study abroad through the Erasmus program. The experience has opened numerous horizons for me and has significantly shaped my academic journey.
I would also like to express my deepest appreciation to the University of Padova, my host institution. This prestigious university has offered me a platform to enrich my knowledge, engage in rigorous research, and gain an international perspective.
I am particularly indebted to my supervisors for their unwavering support and invaluable guidance throughout this project. My sincere thanks to Ph.D. student Anay Deshpande and Professor Federico Chiariotti from the University of Padova, who guided me through the labyrinth of research with their expertise and insights. Their constructive critiques and encouragement have been instrumental in the successful completion of this project.
Additionally, I would like to extend my gratitude to my supervisor from SUP'COM, Ons Ben Rhouma. Her guidance, patience, and support from the preliminary to the concluding level enabled me to develop an understanding of the subject.
Finally, my heartfelt appreciation goes out to my family and friends for their understanding and unwavering faith in me. This journey would not have been possible without their constant encouragement and belief in my abilities.

Contents

List of Figures

List of Tables

Introduction

The Internet of Things is a transformative technological paradigm that has reshaped the landscape of digital communication and computing. In the midst of this revolution, the challenge of energy management looms large, particularly in wake-up radio systems that govern the energy states of IoT devices. These systems decide when a device should be active or in a low-energy state, making them critical to the overall energy efficiency of an IoT network. The burgeoning field of intelligent scheduling provides an intriguing solution to this pressing issue. This thesis explores the application of Reinforcement Learning and Machine Learning, with a focus on the Deep Q-Network model, to optimize intelligent scheduling for wake-up radio systems in IoT networks.

The main objective of this thesis is to examine the feasibility and effectiveness of using Reinforcement Learning and the Deep Q-Network model to achieve intelligent scheduling. This novel approach will be rigorously compared with traditional baseline models to evaluate its capability in optimizing energy consumption, thereby establishing its viability as a superior alternative.

To offer a holistic view, the report is organized into three chapters. The first one provides a robust foundation by discussing the research environment and offering an overview of the theoretical underpinnings of the project, touching on key areas like wake-up radio systems, machine learning, and reinforcement learning. The second chapter transitions from theoretical foundations to practical applications, detailing the design and simulation environment for Wireless Sensor Networks with an emphasis on intelligent scheduling algorithms and models. The third and final chapter presents an analysis of the simulation results, comparing the proposed intelligent scheduling algorithms with baseline models in terms of energy efficiency and overall performance.

By melding rigorous academic investigation with practical experimentation, this thesis aims to fill a critical gap in existing literature. It aspires to push the boundaries of what is currently understood about intelligent scheduling for wake-up radio systems, and by extension, aims to contribute to the overarching goal of making IoT networks more energy-efficient and sustainable.

Chapter 1

Research Environment and Theoretical Foundations

Introduction

Wake-up radio is a technology that allows Internet of Things (IoT) nodes to respond to requests, which can be ID- or content-based (in the former case, the sensor will send its latest reading if its ID is in the request, while in the latter, it will transmit if the data matches the conditions specified in the request message). This type of system allows for interesting scheduling opportunities, particularly if the sensor measurements are correlated: by carefully crafting scheduling requests, the IoT gateway can save energy and improve its estimate of the state of the system.
This project involves the design of scheduled or partially scheduled strategies that achieve these two goals, by using knowledge on the system state and statistics as well as the sensors' battery states and capabilities. The algorithms will then need to be verified by simulation.

1.1 Research Environment

1.1.1 Host University

I was fortunate to have been selected for the Erasmus program for the final year of my engineering degree at SUP'COM, an opportunity that provided me with unique academic and cultural experiences. This program allowed me to venture beyond traditional boundaries and enrich my educational journey in ways I could not have previously imagined.
During the second semester, my journey took an exciting turn when I commenced my research training with the prestigious University of Padova. This experience at one of Europe's oldest and most esteemed institutions marked a pivotal point in my academic career. Being part of an intellectually stimulating environment that thrives on rigorous research and innovative thought, I found myself immersed in a world where cutting-edge ideas are cultivated and knowledge is continually pushed

beyond established horizons.
This narrative sets the stage for the extensive research work I have undertaken, which forms the core of this thesis report. It underscores the intellectual rigor and the international academic exposure that have shaped my research.

1.1.2 Project overview

This project delves into the realm of intelligent scheduling within wake-up radio systems, a crucial aspect influencing the advancement of Internet of Things (IoT) technology. By systematically dissecting the overarching context, identifying the problems, and formulating potential solutions, the project aspires to devise strategies that optimize energy efficiency while maintaining exceptional performance quality within these systems.

1.1.2.1 General Context

- **The Significance of Intelligent Scheduling in IoT**:
  Scheduling algorithms form the backbone of efficient resource management, especially in IoT systems that consist of numerous interconnected devices. Their relevance to this project lies in their potential to optimize the operation of wake-up radio systems, allowing for effective control of sensor activation and improving energy efficiency. Understanding the principles and functionality of intelligent scheduling is crucial to develop strategies that enhance system performance and conserve energy.

- **Wake-Up Radio Systems**:
  wake-up radio systems are integral to IoT environments, due to their low-power and always-on nature, they offer significant potential in improving the energy efficiency of wireless sensor networks. We particularly focus on their operation, the role they play in sensor communication, and how their functionality can be improved with intelligent scheduling. These systems form the core of our research, as our project is designed to optimize their operation, reducing energy consumption while ensuring effective communication. More details in the theoretical part on WUR.

- **Energy Consumption and Efficiency in IOT**:
  Due to the prevalence of IoT devices and their major impact on global energy consumption, energy efficiency is given top importance when designing IoT systems. This is where wake-up radio system scheduling intelligence is useful. We seek to reduce these devices' energy consumption by improving scheduling, so tackling one of the main problems in the IoT space. We dive into ways that can improve energy efficiency as well as elements that affect energy consumption.

1.1.2.2 Problematic

The prevalent issues impacting the successful integration and operation of wake-up radio systems within the Internet of Things (IoT) are basically related to energy efficiency and scheduling challenges.
In the IoT paradigm, devices operate on battery power, and the always-on functionality of wake-up radio systems necessitates continuous energy consumption. This constant power utilization leads to accelerated battery depletion. Though wake-up radio systems individually operate at low power, when considered in the context of large-scale IoT deployments, the cumulative energy consumption is significant. Hence, the challenge of achieving energy efficiency in wake-up radio systems is of paramount importance.
Furthermore, as the scope of IoT devices expands, managing their operation, especially the timing and manner of their activation and communication, becomes a crucial concern. Intelligent scheduling has emerged as a promising approach to tackle this problem. However, designing a smart scheduling mechanism that can effectively manage the functioning of wake-up radio systems remains a daunting task. A comprehensive, intelligent scheduling solution would need to take into account not just the overall state of the system, but also a myriad of factors such as battery states, operational demands, and the prevailing network conditions.
The complexity of these issues outlines the problematic this research work seeks to address.

1.1.2.3 Proposed Solution

In response to the outlined problems, this thesis proposes an intelligent scheduling solution for wake-up radio systems based on Reinforcement Learning (RL) and Deep Q-Networks (DQN). The primary objective of this strategy is to enhance energy efficiency and improve the state estimation of IoT systems.
Our proposed solution is grounded on the premise of IoT nodes' ability to respond to ID- or content-based requests, further enabling the IoT gateway to conserve energy and enhance its estimate of the system state. The concept is to train an intelligent agent, via reinforcement learning and DQN, that can effectively manage and control the operation of the wake-up radio systems in IoT networks. This trained agent is expected to differentiate and distinguish between different system states and events, effectively deciding when and how to activate the wake-up radio systems.
The reinforcement learning approach allows the agent to learn from its experiences, adjusting its actions based on the rewards received from the environment. DQN, a value-based algorithm in reinforcement learning, assists the agent in approximating the optimal policy that determines the actions leading to maximum long-term return. By incorporating these two powerful machine learning techniques, we aim to optimize scheduling decisions, thereby enhancing the overall system performance and energy efficiency.
This proposed solution is anticipated to address the identified challenges and push the boundaries of what is currently achievable in terms of energy efficiency and state

estimation in IoT systems. The subsequent sections of this report will delve deeper into the implementation, simulation, and results of this research project.

1.2 Theoretical Foundations

The world of IoT is dynamically evolving, and wake-up radio systems are at the heart of this technological revolution. Managing energy consumption and developing intelligent scheduling require more than traditional methodologies. The following sections aim to lay down the theoretical background by delving into critical areas such as wake-up radio systems' fundamentals, machine learning, and reinforcement learning. The journey through these areas will help in understanding how they can be fused to form a more comprehensive approach towards energy efficiency and intelligent management in IoT.

1.3 Wake-Up Radio Systems in IoT Networks

Wake-up radio systems have become essential to the Internet of Things (IoT) as energy conservation remains a critical constraint. The fundamental idea behind this technology is the use of a low-power, secondary radio that remains in a 'listening' state to monitor a special "wake-up" signal. Upon detecting this signal, the primary high-power radio system is triggered to "wake up" from its low-power state to perform essential functions.

Architecture The architecture of wake-up radio systems often consists of two components:

1. **Primary Radio**: Responsible for the main communication tasks but stays in a low-power sleep mode to conserve energy when not active.

2. **Secondary Radio**: Operates at a much lower power and constantly listens for the wake-up signal.

Communication Protocols Wake-up radio systems typically use specialized communication protocols optimized for low power consumption and latency. These protocols ensure seamless handover between the secondary and primary radios without compromising on speed and efficiency.

Reducing False Wake-Ups To minimize energy wastage and improve reliability, advanced algorithms and filtering mechanisms are employed to reduce false wake-ups, which occur when the secondary radio misinterprets noise or irrelevant signals as a wake-up command.

Applications and Benefits Wake-up radio systems find applications across various industry sectors including manufacturing, healthcare, and smart cities. The primary benefit is significant energy savings, leading to longer device lifetimes and reduced operational costs.

The Need for DRL While traditional methods for managing wake-up radio systems are effective to some extent, they often lack the ability to adapt to dynamic network conditions. This is where Deep Reinforcement Learning (DRL) comes into play. By using DRL, we can develop adaptive, data-driven algorithms to optimize the wake-up signal's timing and other operational parameters.

1.4 Intelligent Scheduling

Intelligent scheduling is a cornerstone technology that aims to enhance the efficiency of task management within computer and network systems. In wake-up radio systems, intelligent scheduling plays a vital role in determining the optimal timing and conditions for dispatching wake-up signals. The goal is to strike an optimal balance between energy conservation, system performance, and latency, while also being flexible enough to adapt to real-time changes in network conditions.

Heuristic Methods Heuristic methods are rule-based algorithms that provide good-enough solutions in a reasonable time frame. For example, in wake-up radio systems, a simple heuristic might involve sending wake-up signals at fixed intervals or in response to certain predefined events. Although heuristic methods are computationally less intensive, they often lack the ability to adapt to changing conditions.

Machine Learning Approaches Machine learning-based intelligent scheduling involves the use of algorithms that can learn from data. These algorithms can be trained to recognize patterns and make predictions, thereby optimizing the wake-up schedules more efficiently. However, these methods often require a significant amount of data and computational resources for training.

Optimization Techniques Optimization techniques such as linear programming or genetic algorithms aim to find the best possible solution from a set of feasible solutions. In the context of wake-up radio systems, these techniques can calculate the optimal timings for wake-up signals based on a set of constraints like energy limits, latency, and signal reliability.

Deep Reinforcement Learning Deep Reinforcement Learning (DRL) is increasingly being used in intelligent scheduling for wake-up radio systems. It combines neural network-based function approximators with reinforcement learning algorithms to optimize wake-up schedules. The agent in a DRL framework, typically the secondary low-power radio, interacts with the environment, i.e., the network, to receive rewards based on how well it schedules the wake-up of the primary radio.

Adapting to Real-Time Changes One of the key advantages of intelligent scheduling algorithms, particularly those based on machine learning and optimization techniques, is their ability to adapt to real-time changes in network conditions. For example, if the network experiences an increase in data traffic, the intelligent scheduling algorithms can dynamically adjust the wake-up signal timings to maintain optimal performance and energy efficiency.

Finally Intelligent scheduling is fundamental to the effective functioning of wake-up radio systems. By leveraging various algorithmic approaches, it is possible to tailor the wake-up signal timings to diverse scenarios and requirements, thereby significantly improving system performance, reducing latency, and conserving energy.

1.5 Machine Learning

Machine Learning is a subfield of Artificial Intelligence (AI) that provides systems the ability to automatically learn from data and improve performance without being explicitly programmed. Unlike deep learning, which is a subset of machine learning specializing in deep neural networks, general machine learning covers a broad range of algorithms from linear regression to decision trees, support vector machines, and simple neural networks.
Machine Learning has been applied in various domains such as natural language processing, medical diagnosis, and financial forecasting. It is particularly effective in scenarios where the explicit algorithmic solution is difficult to formulate.

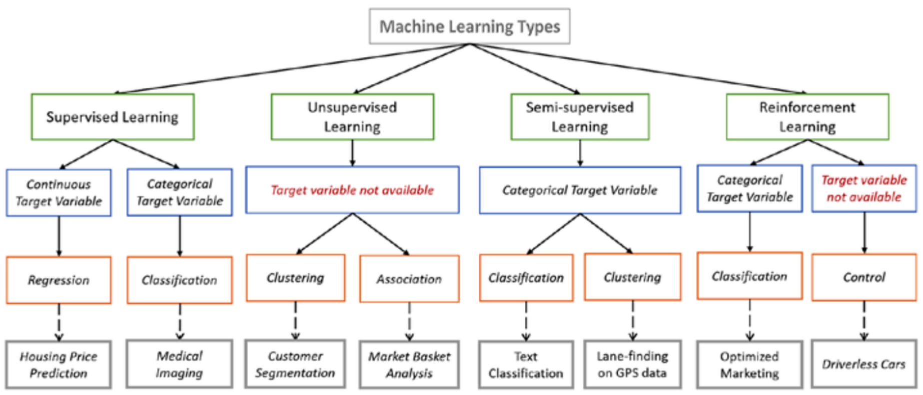

Figure 1.1: Machine Learning Algorithm Architecture

The structure shown in Figure 1.1 indicates the diversity of algorithms that come under the umbrella of machine learning. These algorithms can vary in complexity and are chosen based on the problem at hand, the nature of the input and output data, and the type of model to be learned.
In the context of this thesis, machine learning is crucial for the development and implementation of Q-Learning-based intelligent scheduling in wake-up radio systems. Unlike deep learning models, the Q-Learning model implemented here is less computationally intensive, making it well-suited for resource-constrained environments like IoT systems.

1.5.1 Neural Networks

- **Structure of Neural Networks**:
  Neural Networks are the foundational building blocks of deep learning, inspired by the human brain's structure and functioning. A Neural Network consists of interconnected layers of artificial neurons or nodes as shown in the Figure 1.2, typically organized into three main categories:

  - **Input Layer**: This is where the network receives its input data. Each neuron in this layer represents a unique feature or attribute of the input.
  - **Hidden Layers**: These layers are positioned between the input and output layers and contain neurons that process the input data. The computations and transformations within hidden layers are where the "learning" occurs.
  - **Output Layer**: This final layer produces the prediction or classification, translating the processed information from the hidden layers into a format that serves the specific task, such as a probability distribution over different classes.

  Each connection between neurons is associated with a weight, and each neuron has a bias. These weights and biases are the parameters that the network learns through training.

- **Feedforward Neural Networks**:
  Feedforward Neural Networks are the simplest type of neural network architecture, where information moves in one direction from the input layer through the hidden layers to the output layer, without any loops. This unidirectional flow is what defines them as "feedforward."

- **Activation Functions**:
  Activation functions are mathematical equations that determine the output of a neural network. They introduce non-linear properties into the system, allowing the network to learn complex patterns. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU).

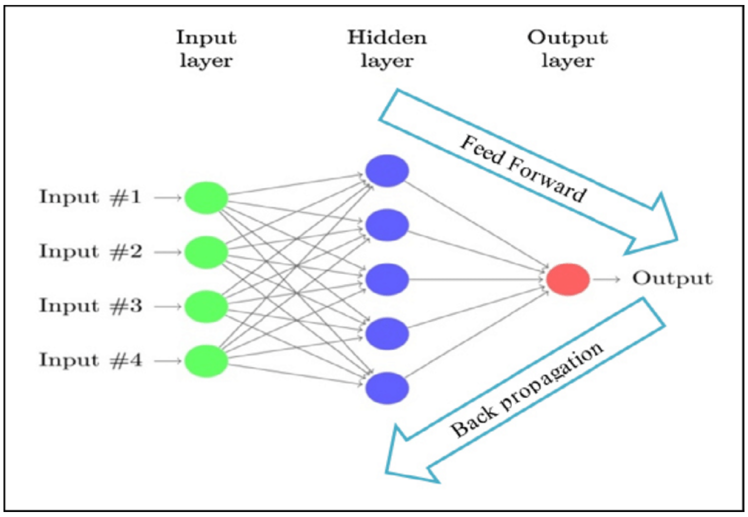

Figure 1.2: Structure of a Neural Network

- **Backpropagation Algorithm**:
  Backpropagation is a crucial algorithm used to train neural networks. It calculates the gradient of the loss function concerning each weight by applying the chain rule, effectively "propagating" the gradients back through the network. By adjusting the weights and biases in the direction that minimizes the loss, the network "learns" to make more accurate predictions.

1.5.2 Training and Optimization

Training and optimization are fundamental building blocks in the field of machine learning. They lay the groundwork for constructing robust models capable of complex predictions and intelligent decision-making. In the context of wake-up radio systems, these concepts become paramount as they enable the creation of efficient algorithms for intelligent scheduling. This chapter delves into the essential components of training and optimization, shedding light on their relevance to the development of advanced scheduling solutions in IoT applications.

- **Loss functions**:
  Loss functions quantify the difference between the predicted outcomes and the actual results. They are the guiding metrics that machine learning models strive to minimize. For intelligent scheduling in wake-up radio systems, an appropriate loss function would be pivotal in steering the model towards accurate predictions, leading to energy-efficient and well-informed scheduling decisions.

- **Optimization Algorithms**:
  Optimization algorithms dictate how a model learns by adjusting its internal parameters. These algorithms guide the way the model traverses the error

landscape, seeking the optimal configuration. In the domain of wake-up radio systems, the choice of an optimization algorithm can heavily impact the efficiency and effectiveness of the intelligent scheduling process.

- **Regularization Techniques**:
  Regularization is the art of preventing overfitting, ensuring that the model generalizes well to unseen data. In the context of intelligent scheduling, implementing robust regularization techniques helps in building a model that doesn't just memorize the training data but understands the underlying patterns, enabling reliable scheduling decisions within diverse wake-up radio environments.

- **Hyperparameter Tuning**:
  Hyperparameters are parameters that are set prior to the training process, and they play a crucial role in defining the model's architecture and training dynamics. Fine-tuning these hyperparameters can lead to significant improvements in the model's performance. For intelligent scheduling, optimal hyperparameter selection translates into more efficient learning, which is vital for making real-time scheduling decisions in wake-up radio systems.

1.6 Reinforcement Learning

Reinforcement learning entails learning how to take optimal actions in various scenarios to maximize a given numerical reward. The learning agent is not provided with predefined actions to execute but must independently determine which actions result in the highest rewards through trial and error. One of the challenges is that certain actions may impact not only immediate rewards but also future scenarios, thus affecting long-term rewards. The two key elements that set reinforcement learning apart are the necessity for trial-and-error and the concept of delayed rewards.
The reinforcement learning problem is mathematically modeled utilizing concepts from dynamical systems theory, specifically focusing on the optimal management of Markov decision processes with incomplete information. We will delve into the technical specifics in upcoming sections, but the overarching objective is to model the critical components that a learning agent encounters when engaging with its environment over time to achieve a goal. The model captures three basic aspects—sensing the environment, taking actions, and setting goals. Any methodology effectively addressing these elements is considered part of the reinforcement learning domain.
Reinforcement learning differs from unsupervised learning, which primarily aims to uncover hidden patterns in unlabeled data sets. While it may seem logical to categorize reinforcement learning as a form of unsupervised learning due to its absence of explicitly correct behavioral examples, the focus in reinforcement learning is on maximizing a reward metric rather than identifying hidden structures. Therefore, we argue that reinforcement learning constitutes a separate paradigm in machine learning, distinct from supervised and unsupervised learning methods.

Lastly, reinforcement learning aligns with a broader shift back towards fundamental principles in artificial intelligence (AI). During the late 20th century, the common belief was that intelligence was the result of numerous specialized techniques and heuristics rather than universal principles. The notion was that intelligence would emerge from machines loaded with sufficient data, say millions or billions of facts. Methods relying on general principles were labeled as "weak," while specialized techniques were considered "strong." However, contemporary AI research has shifted towards identifying universal principles governing learning, decision-making, and search. While the extent of this shift remains to be seen, it is evident that research in reinforcement learning is a part of this broader trend back to basic principles of AI.

1.6.1 Some Examples

To grasp the essence of reinforcement learning, it's beneficial to consider various instances and potential use-cases that have inspired its progress.

- An expert in chess selects a game move. This decision is guided both by strategic foresight—considering potential responses and counter-responses—and by instinctual evaluations of the value of different board positions and potential moves.
- A dynamic controller fine-tunes the operational parameters of an oil refinery in real-time. The controller aims to balance the trade-offs between output, cost, and quality, deviating from the initial guidelines provided by engineers to meet specific marginal cost criteria.
- A newborn gazelle manages to stand up just moments after its birth. Within the next half hour, it accelerates to speeds of 20 miles per hour.
- An autonomous cleaning robot deliberates whether to explore a new room for additional waste or to start navigating back to its charging station. The robot's decision hinges on its current battery status and its past experiences in locating the charging station.

In each of these scenarios, the agent utilizes its accumulated experience to enhance its future performance. The chess expert fine-tunes the intuitive judgments used to assess board states, thereby advancing his gameplay. Similarly, the gazelle calf elevates its running efficiency over time. Pre-existing knowledge—whether acquired from related tasks or innate capabilities—shapes what is convenient or beneficial to learn. However, active interaction with the surrounding environment remains crucial for tailoring behavior to meet specific task requirements.

1.6.2 Trade off between Exploration and Exploitation

A unique hurdle in the realm of reinforcement learning is the balancing act between exploration and exploitation. To maximize its rewards, the agent should lean towards actions that have previously yielded positive outcomes. However, to unearth these rewarding actions, the agent also needs to venture into uncharted territory by choosing actions it hasn't taken before. This creates a dichotomy: the agent needs to utilize its existing knowledge for immediate gains, yet also engage in new experiences to refine future choices. The conundrum lies in the fact that an exclusive focus on either exploration or exploitation could result in task failure. Therefore, the agent needs to adopt a diversified approach, incrementally favoring actions that seem promising based on past experience. In tasks with stochastic outcomes, an action must be sampled multiple times to acquire a reliable understanding of its reward probability. While the exploration-exploitation conundrum has been a subject of extensive mathematical scrutiny for years, a definitive solution remains elusive. Importantly, this specific issue of juggling between exploration and exploitation is largely absent in traditional supervised and unsupervised learning paradigms.

1.6.3 Epsilon-Greedy

A simple approach to deal with exploration is ϵ -Greedy Policy, that is a stochastic policy:

$$A_t = \begin{cases} \text{greedy action with probability } 1-\epsilon \\ \text{non greedy action with probability } \epsilon \end{cases} \tag{1.1}$$

with $\epsilon \in (0,1)$ (typically a small number).
- The non greedy actions are chosen with uniform probability.

1.6.4 Markov Decision Processes

Markov Decision Processes (MDPs) provide a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker or agent. MDPs are particularly useful in the context of sequential decision-making problems where an agent must interact with an environment to achieve a goal.
An MDP is defined by the tuple (S, A, P, R):

- **States (S):** A finite set of states that the process can be in at any given time.
- **Actions (A):** A finite set of actions that the agent can take.
- **Transition Probabilities (P):** A function $P(s'|s,a)$ that defines the probability of transitioning from state s to state s' after taking action a.
- **Reward Function (R):** A function $R(s,a,s')$ that defines the immediate reward received by the agent after transitioning from state s to state s' by taking action a.

A policy π maps states to actions, defining the behavior of the agent. The goal in many MDP problems is to find an optimal policy π^* that maximizes the expected cumulative reward over time, often discounted by a factor γ.

MDPs assume the Markov property, meaning that the future state depends only on the current state and action and not on the sequence of states that preceded it.
In the context of wake-up radio systems and intelligent scheduling, the application of MDPs enables modeling the interactions between the IoT gateway and the nodes. Actions might include sending scheduling requests, states could represent the system's state and nodes positions, and rewards might Age of Information and system performance. Utilizing MDPs, along with Reinforcement Learning and DQN, allows for the development of intelligent scheduling strategies that adapt to the dynamic nature of the system, thereby optimizing energy consumption and system performance.

1.6.4.1 The Agent Environment Interface

The fundamental idea behind Markov decision processes (MDPs) is to provide a clear framework for understanding how to achieve objectives through interactive learning. In this framework, the entity that learns and makes choices is referred to as the 'agent,' while the external factors that the agent interacts with are collectively termed the 'environment.' This interaction is ongoing, with the agent deciding on actions and the environment reacting to those decisions, thereby presenting new scenarios to the agent, as illustrated in Figure 1.3. The environment also generates numerical rewards that the agent aims to maximize over a series of actions.

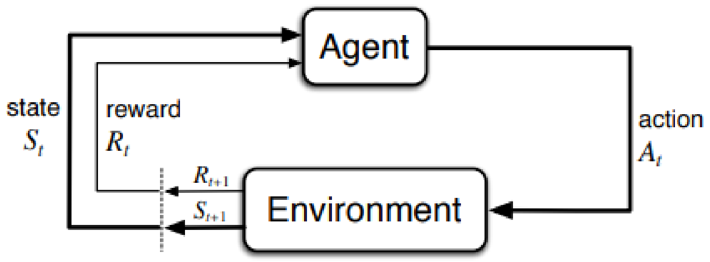

Figure 1.3: Markov decision process agent–environment interaction.

1.6.4.2 Rewards and Goals

In the context of reinforcement learning, the agent's objectives are encoded through a specific numerical signal, commonly known as the 'reward,' which is conveyed from the environment to the agent. At any given point in time, this reward is a numerical value, denoted as $R_t \in \ R$. The informal aim of the agent is to optimize the cumulative sum of the rewards it gathers, focusing on long-term gains rather than immediate rewards. This idea can be formally articulated through the 'reward hypothesis':

> The essence of any goal or purpose can be adequately expressed as the optimization of the expected value of a cumu-

lative scalar signal, referred to as the 'reward.'

The incorporation of a reward signal as the defining measure of an objective is a distinguishing attribute of reinforcement learning.
Though the use of rewards as goals may initially seem restrictive, it has demonstrated remarkable adaptability and broad applicability. To illustrate, consider various scenarios where rewards have been effectively utilized: To teach a robot how to walk, a reward corresponding to its forward movement at each time step is offered. To train a robot in maze navigation, a reward of -1 is given for every time step taken before successfully exiting the maze, encouraging quick escapes. In a recycling task, a robot could be given zero reward most of the time and a +1 reward for each can collected. In games like checkers or chess, the most intuitive rewards would be +1 for a win, -1 for a loss, and 0 for a draw or for positions that are not yet concluded. The key point across these instances is that the agent is trained to maximize its reward. To align the agent's behavior with our objectives, the reward structure must be crafted in a manner that naturally leads the agent towards our intended goals.

1.6.4.3 Episodes and Returns

The agent's objective is to optimize the sum of rewards over an extended period. How can we state this formally? If we represent the sequence of rewards obtained after the time step t as $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$, what specific property of this series are we aiming to optimize? Generally, the target is to maximize the 'expected return,' where the return G_t is a particular function of the series of rewards. In its most basic form, the return is simply the aggregate of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \tag{1.2}$$

Here, T stands for the concluding time step. This framework is applicable in scenarios where the interaction between the agent and the environment naturally divides into distinct sequences, or 'episodes,' like game plays, maze navigations, or any cyclical interactions. Each episode terminates at a unique state, known as the 'terminal state,' before the system resets to a standard initial state or a state sampled from a predetermined distribution.

An additional notion to consider is 'discounting.' In this method, the agent aims to perform actions that maximize the sum of future rewards, weighted by a discount factor. Specifically, it selects A_t to optimize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.3}$$

Here, γ, ranging between 0 and 1, is the 'discount rate.'
The discount rate calculates the current value of future rewards; a reward obtained k steps in the future is considered to be worth γ^k times its immediate value.
When $\gamma = 0$, the agent becomes 'short-sighted,' focusing solely on immediate rewards.

As γ nears 1, the agent increasingly takes future rewards into account, becoming more 'long-sighted.'

1.6.4.4 Value Functions and Policies

- **Policy:** The agent's behaviour function, a map from state to action.
- **Value funtion:** Prediction of future rewards from a state $V_\pi(s)$
- **Q-Value funtion:** Prediction of future rewards from a (state, action) pair $q_\pi(s, a)$.

The RL goal is to find the optimal policy(the one maximizing the cumulative rewards).
We can have:
- Deterministic policies: a $= \pi(s)$
- Stochastics policies: $\pi(a|s) = P(A_t = a|S_t = s)$
- Model of the environment can be used by the agent (if available).
- We can have models that predict the next state and others that predict the next immediate rewards.
- We can find model-free approaches as opposed to model-based approaches as described in the figure 1.4.

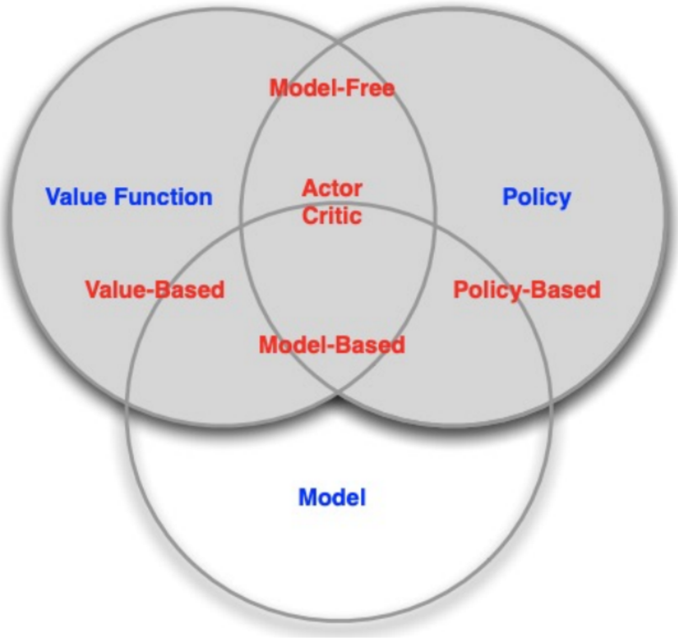

Figure 1.4: Components of an agent

1.6.4.5 On-Off Policy

On-policy learning is a method where the learning agent learns the value of the policy being carried out by the agent itself. In this approach, the policy used to make decisions is the same policy that is improved and evaluated. It focuses on improving the current policy that the agent is following. A well-known on-policy algorithm is SARSA.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1.4}$$

where α is the learning rate, γ is the discount factor, and Q represents the action-value function.
Contrarily, off-policy learning allows the agent to learn a different policy from the one it follows. The policy used to make decisions (behavior policy) and the policy that is improved and evaluated (target policy) can be different. This enables the agent to explore and learn from actions that are not part of its current policy. The Q-learning algorithm is a typical example of off-policy learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{1.5}$$

The main distinction between on-policy and off-policy methods lies in their approach to exploration and exploitation. On-policy methods are more conservative, as they strictly follow the current policy. Off-policy methods, on the other hand, allow the agent to learn from exploratory actions, potentially leading to a more optimal policy.
In the context of this work, the choice between on-policy and off-policy methods depends on the specific requirements of the intelligent scheduling task in wake-up radio systems and the trade-offs between exploration and exploitation.

1.6.4.6 Optimality and Approximation

Optimal Policies
An optimal policy is a policy that achieves the maximum possible long-term reward for an agent interacting with an environment. Formally, a policy π^* is said to be optimal if it attains the highest expected cumulative reward among all possible policies:

$$\pi^* = \arg\max_\pi V_\pi(s), \quad \forall s \in S \tag{1.6}$$

where $V_\pi(s)$ represents the value of following policy π from state s, and S is the set of all possible states.

Optimal Value Functions
The optimal value function correspond to the optimal policy and provide a way to measure the quality of states and actions. There are two main types:

- **Optimal State-Value Function:** Represents the expected cumulative reward when following the optimal policy from a given state s:

$$V^*(s) = \max_{\pi} V_\pi(s), \quad \forall s \in S \tag{1.7}$$

- **Optimal Action-Value Function:** Represents the expected cumulative reward when taking action a in state s and thereafter following the optimal policy:

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a), \quad \forall s \in S, a \in A \tag{1.8}$$

In many real-world problems, finding exact optimal policies and value functions may be intractable. Approximation methods aim to find policies and value functions that are close to optimal but computationally feasible. Techniques like function approximation and algorithms like Deep Q-Networks (DQNs) can be employed to approximate the optimal policy and value functions.

In the context of intelligent scheduling in wake-up radio systems, optimality plays a crucial role in defining the best policies for energy efficiency. Approximation methods allow tackling the complexity of the environment and finding near-optimal solutions that balance energy consumption and system performance.

1.7 Deep Reinforcement Learning

Deep reinforcement learning (DRL) is an advanced area of reinforcement learning that utilizes deep learning techniques to handle complex, high-dimensional state spaces. It has found applications in various domains, including gaming, robotics, and intelligent scheduling in IoT, as in wake-up radio systems.

1.7.1 Q-Learning

Q-learning is a model-free reinforcement learning algorithm that seeks to find the optimal action to take in a given state. The Q-learning update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \tag{1.9}$$

where:

- s and a are the current state and action.
- s' and a' are the next state and next action.
- r is the reward received.
- α is the learning rate.

- γ is the discount factor.

Q-learning directly learns the optimal policy by iteratively updating the Q-values for each state-action pair.

1.7.2 Deep Q-Networks (DQN)

Traditional Q-learning struggles with large state spaces, leading to the development of DQN. Deep Q-Networks use neural networks to approximate the Q-value function:

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{1.10}$$

where θ represents the parameters of the neural network.

DQN combines the power of Neural Networks with Q-learning, allowing the agent to deal with high-dimensional input spaces. Key features of DQN include experience replay and target networks, which stabilize training and enhance convergence.
In traditional Q-learning, the Q function is often stored in a table, but this becomes impractical for large or continuous state spaces. DQNs replace this table with a neural network as represented in the figure 1.5. it takes the state as input and outputs a Q value for each possible action. This allows DQNs to handle more complex environments with higher-dimensional state spaces. By combining Neural Network's ability to process complex data with Q-learning's strategy for learning optimal actions, DQNs offer a powerful approach to tackle more intricate and challenging problems.

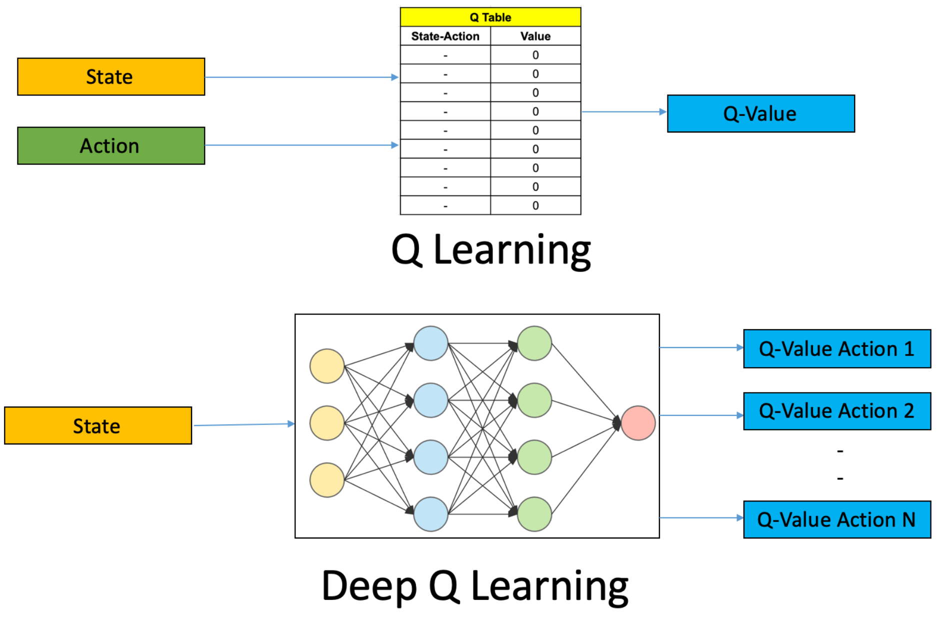

Figure 1.5: From Q-learning to DQN

1.7.3 Relevance to Wake-Up Radio Systems

Wake-up radio systems play a critical role in optimizing energy consumption in wireless networks, especially IoT. These systems utilize a low-power radio to listen for a "wake-up" signal, activating the primary radio system only when needed. While this architecture significantly reduces energy consumption, optimizing the scheduling of these wake-up signals is a complex problem that involves various parameters such as latency, energy efficiency, and overall system performance.

Deep Reinforcement Learning (DRL), with algorithms like Q-Learning and Deep Q-Networks (DQN), offers an innovative approach to tackle this problem. Unlike traditional methods, which may rely on fixed heuristics or manual tuning, DRL provides a flexible, data-driven way to optimize system behavior. The agent in a DRL model learns to make decisions by interacting with the environment, receiving feedback in the form of rewards or penalties based on the actions taken. This ability to learn from the environment makes DRL highly adaptable to the dynamic conditions often found in wireless networks.

In the case of wake-up radio systems, a DRL model could be trained to decide the optimal times to send wake-up signals, considering various metrics like signal strength, battery status, and network traffic. The DRL model can be designed to weigh the importance of these metrics differently, depending on whether the system prioritizes energy efficiency, latency, or a balance of both. This leads to a more intelligent scheduling strategy that adapts to real-time conditions, thereby maximizing the overall performance and reliability of the wake-up radio system.

Moreover, the scalability of DRL algorithms allows them to be applied in large, complex network architectures. As IoT networks continue to grow and evolve, employing DRL for intelligent scheduling in wake-up radio systems can offer significant advantages in adaptability and efficiency, making it a highly relevant and promising approach for future advancements in this domain.

Conclusion

This concludes the initial chapter of the thesis, which serves as a comprehensive guide to both the research environment and the theoretical groundwork that underpins this investigation. We have explored the critical aspects of wake-up radio systems in the IoT landscape, specifically focusing on the challenges they present in terms of energy efficiency and intelligent scheduling. Reinforcement Learning and Deep Q-Networks have been presented as promising tools to optimize these systems and address the associated issues.

In addition to laying the contextual foundation, we have traversed the theoretical landscape that informs our approach, delving into key concepts related to IoT, wake-up radio systems, energy efficiency, and intelligent scheduling. We have also provided an in-depth introduction to Reinforcement Learning and DQN, highlighting their relevance and application in our proposed framework. This theoretical exploration not only enhances our understanding of the problem but also sets the stage for the subsequent practical implementations.

As we move forward, we will apply these theoretical insights to the practical design and simulation of intelligent wake-up radio systems. This transition from theory to practice marks a logical and necessary step in our research journey, symbolizing the evolution from conceptual understanding to actionable solutions.

Chapter 2

WSN Environment Design and Simulation

Introduction

The rapid development of IoT has resulted in an increased interest in Wireless Sensor Networks (WSNs). Sensor networks are fundamental to numerous applications such as environmental monitoring, healthcare, smart cities, and industrial automation. Managing these networks efficiently requires intelligent strategies for sensor placement, event detection, scheduling, and communication. This chapter introduces a novel approach to addressing these challenges through the development and simulation of a customized WSN environment.

Objectives: The primary objectives of this chapter are:

- To define a WSN environment using the Gym framework, incorporating elements such as sensor placement, event generation, and reward structure.
- To implement various scheduling algorithms, including customized Q-learning, DQN, Round Robin, and Random Scheduler, to compare their effectiveness.
- To evaluate and analyze the results, uncovering insights into the performance, challenges, and opportunities within the chosen methodologies.

Structure of the Chapter: The remainder of this chapter is organized as follows:

- **Section 3.1** delves into the design details of the WSN environment, defining the spatial layout, event mechanics, and the attributes of observation and action spaces.
- **Section 3.2** covers the simulation of the environment using various models and algorithms, highlighting the implementation process and concludes the chapter with a summary and final remarks.

2.1 WSN Environment Design

2.1.1 Technologies Used

In the development of the WSN environment, a range of technologies and tools were employed to ensure effective implementation, simulation, and analysis. The following subsections highlight the primary technologies leveraged.

2.1.1.1 Gymnasium

The OpenAI Gym library is a popular toolkit in the field of reinforcement learning, providing standardized environments for developing and testing algorithms. It offers a diverse suite of predefined environments, ranging from simple control tasks to game playing, and allows users to create custom environments with a standardized interface. Key components include defining observation and action spaces, step and reset functions, a customizable reward system, and various rendering modes for visualization. In the context of this project, OpenAI Gym was utilized to structure the Wireless Sensor Network (WSN) environment, leveraging its features and flexibility to create a unique simulation environment. Its open-source nature and compatibility with other machine learning libraries like TensorFlow and PyTorch further enhance its utility and extensibility.

Figure 2.1: Gym: An API standard for reinforcement learning

2.1.1.2 Stable Baselines 3 (SB3)

Stable Baselines3 is a set of high-quality implementations of reinforcement learning algorithms in Python. It builds upon the foundations of Stable Baselines and OpenAI's Baselines, providing a clean and simple interface that's consistent with OpenAI Gym. The primary goal of SB3 is to enable efficient experimentation and development by providing well-tested and well-documented code. The library includes a variety of advanced algorithms like Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Twin Delayed DDPG (TD3), among others.

Stable Baselines 3 is built on top of PyTorch, allowing seamless integration with deep learning workflows. It fosters research reproducibility by adhering to a strict coding standard and providing comprehensive test coverage. In this project, Stable Baselines 3 was leveraged to benchmark the performance of the custom-designed DQN against standard models, benefiting from its ease of use, flexibility, and extensive feature set.

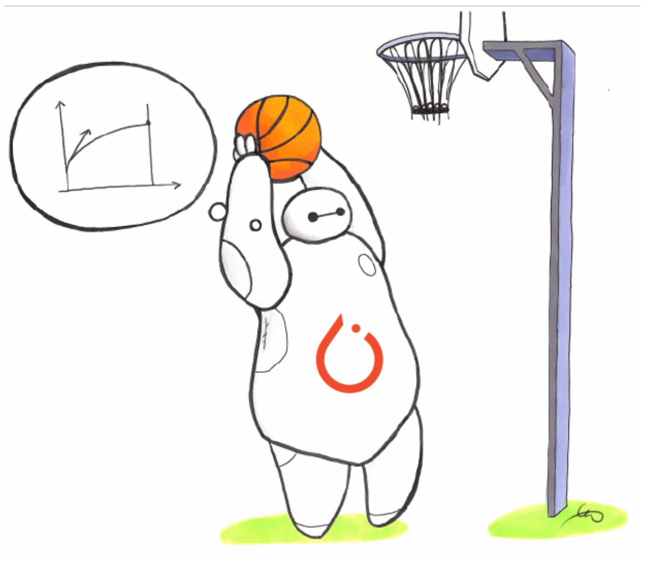

Figure 2.2: Stable Baselines 3

2.1.1.3 Additional Tools and Libraries

The project was implemented using Python, a powerful and versatile programming language renowned for its ease of use and extensive libraries in scientific computing and machine learning. In addition to Python, a variety of tools and libraries were utilized to enhance the functionality, efficiency, and ease of development for the WSN environment. These include:

- **NumPy:** A fundamental package for scientific computing with Python, offering support for large, multi-dimensional arrays and matrices.
- **PyTorch:** An open-source machine learning framework used for constructing deep neural network models, offering tensor computations with GPU acceleration.
- **SciPy:** An ecosystem for mathematics, science, and engineering, including statistical functions and other specialized tools.
- **Matplotlib:** A 2D plotting library for creating static, interactive, and animated visualizations in Python.
- **tqdm:** A fast and extensible progress bar used to provide visual feedback during lengthy computations.
- **MessagePack (msgpack):** A binary serialization format used to save model parameters and support NumPy arrays.
- **TensorBoard:** A visualization toolkit for TensorFlow, used in conjunction with PyTorch for tracking and visualizing metrics such as loss and accuracy.

These tools offered essential capabilities for numerical computation, data visualization, deep learning, serialization, progress tracking, and other tasks pivotal to the project's success.

2.1.2 Design of the WSN Environment

The system model for the wireless sensor network deployed in a hostile region is detailed below. The sensor nodes are strategically placed to cover the entire region while maintaining network connectivity as illustrated in Figure 2.3. Each sensor node has a circular coverage area with a specified radius. The deployment of sensor nodes in the hostile region follows a Poisson point process distribution, ensuring a random and even spatial distribution (both random and evenly spaced throughout the hostile region. The use of a Poisson point process for deployment ensures a random spatial distribution of the sensors, while maintaining a relatively uniform spacing between them. This distribution helps to provide coverage across the entire region and ensures that the sensors are not concentrated in specific areas).
Events are generated within the hostile region uniformly using a Bernoulli random variable. Each event is binary, taking on values of either None or 1. During the pulling process, the gateway sequentially pulls updates from the sensor nodes depending on the type of scheduling or the algorithm (Round-Robin, Random, RL-agent,..).
To determine the generated events, a threshold probability, denoted as p_{th}, is introduced. When an event is generated within the hostile region, its probability of being occurred is evaluated. If the event's probability is lower than the threshold probability ($p < p_{th}$), it is considered a non-occurred event and assigned a None. Conversely, if the event's probability exceeds the threshold ($p \geq p_{th}$), it is deemed an occurred event and assigned a value of 1.
By incorporating the threshold probability, the system allows for selective event occurrence based on the event's probability. This provides the flexibility to adjust the threshold probability and optimize the system's performance in terms of event occurrence, detection sensitivity and false positives.

- **Sensor Placement and Coverage:** Sensors are deployed according to a Poisson point process distribution in a 2D space. Each sensor has 2 coordinates indicating its position, with a circular coverage radius, customizable based on the object. The hostile region has a total surface area of 1, ensuring comprehensive coverage.

- **Event Generation and Detection:** Events are generated based on a random variable exceeding a threshold probability. If an event occurs within a sensor's coverage area, it is placed in the sensor's buffer. During an action, the selected sensor's buffer is emptied, and the event is removed from all other buffers. If the event probability is less than the threshold, it is considered None.

- **Observation and Action Spaces:** The observation space is an Nx3 matrix, with N representing the number of sensors. The first two columns indicate

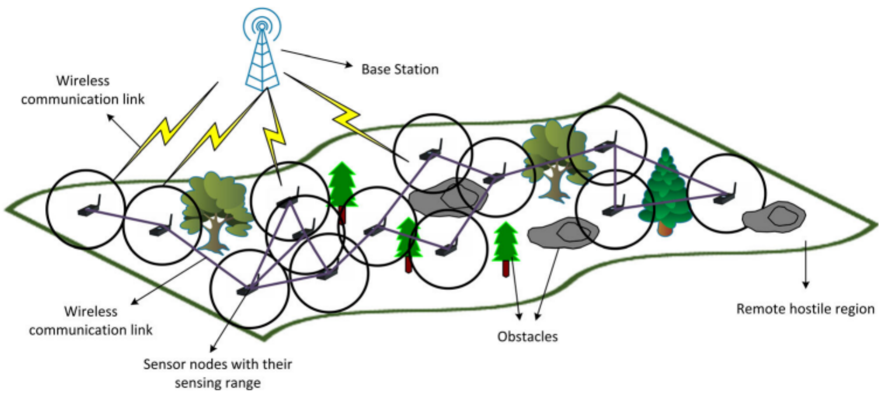
Base Station
Wireless communication link
Wireless communication link
Sensor nodes with their sensing range
Obstacles
Remote hostile region

Figure 2.3: Wireless Sensor Network model

sensor positions, while the third column tracks sensor selection count during the episode. The action space consists of the sensor indices, ranging from 0 to N-1.

- **Reward Structure:** The reward function considers event type and selected action. For a None event, the reward is 0. If the event is not None and the selected sensor's buffer is empty, a penalty of -0.4 is applied. Otherwise, the reward is calculated as:

$$\text{reward} = \sum_{x \in \text{selected\_buffer}} \frac{1}{\text{AoI}}$$

  where AoI (Age of Information) represents the time since the event was generated.

- **Simulation Parameters:** The default parameters for the simulation are set as:

```
sensor_numbers=13, sensor_coverage=0.4, max_steps=1000, threshold_prob=0.3
```

  These can be customized to suit specific requirements or scenarios.

2.2 WSN Environment Simulation

The simulation is carried out to validate and analyze the performance of the proposed Wireless Sensor Network (WSN) design in the hostile environment. The following subsections detail the components and steps involved in the simulation.

2.2.1 Pipelines

- **Generate the Sensor Nodes:** Deploy the sensor nodes in the hostile region following a Poisson point process distribution. This ensures a random spatial distribution of the sensor nodes.
- **Define the Coverage Areas:** Assign circular coverage areas to each sensor node. These areas represent the regions within which a sensor can detect and capture events.
- **Generate Events:** Utilize a Bernoulli random variable to generate events uniformly within the hostile region. Depending on a threshold probability p_{th}, each event takes a value of 1 or None. Specifically, if the random variable is greater than or equal to p_{th}, the event is considered "occurred" and assigned a "1". Otherwise, it is assigned None.
- **Evaluate Event-Sensor Distance:** If the event is assigned a value of 1, calculate the distance between the event and all sensors. If this distance is less than the coverage radius, the event is added to the corresponding sensor's buffer.
- **Select an Action:** Choose an action based on the scheduling algorithm being used, such as Round-Robin, Random, RL-agent, etc.
- **Update After Action Selection:** Perform the necessary updates to the environment after selecting a specific sensor, such as updating buffers and rewards.

2.2.2 Models and Algorithms

2.2.2.1 Round Robin Scheduler

The Round Robin scheduling algorithm is implemented as a simple loop within the simulation environment. it was executed for a specified number of episodes and the scores for each episode was collected.

Details of the Implementation

1. **Number of Episodes**: The algorithm is run for a defined number of episodes, in this case, 20.
2. **Initialization**: At the beginning of each episode, the environment is reset to its initial state, and variables to track the score and current index are initialized.
3. **Action Selection**: The Round Robin scheduler simply cycles through the available actions, in this case, denoted by the index i. The index is incremented at each step and wrapped around once it reaches a specified limit RR, which is equal to the number of points in the environment.

4. **Step Execution**: At each step, the selected action is executed within the environment using the `env.step` method. The reward is added to the episode's score, and the termination conditions are checked.

5. **Termination**: The loop continues until a termination or truncation condition is met, marking the end of the episode. The final score for the episode is recorded.

6. **Results**: The scores for all episodes are printed, along with the average score across the episodes.

Operation in the Context of the Simulation

The Round Robin scheduler is a simple and deterministic approach, where each action or task is selected in a cyclic order. In the context of the simulation:

- **Fairness**: By cycling through all possible actions, the Round Robin scheduler ensures that each action is selected with equal frequency over time.
- **Simplicity**: The implementation is straightforward and does not require any complex logic or data structures.
- **Predictability**: Since actions are selected in a fixed order, the behavior of the scheduler is highly predictable.
- **Limitations**: The Round Robin scheduler does not take into account any specific characteristics or requirements of the actions, such as priority or complexity. This may lead to suboptimal performance in some scenarios.

In summary, the Round Robin scheduler provides a simple and fair approach to action selection in the simulation. While it may not be the most efficient or sophisticated method, it serves as a useful baseline for comparison with more complex scheduling algorithms.

2.2.2.2 Random Scheduler

The Random scheduling algorithm is implemented as a loop within the simulation environment. it was executed for a specified number of episodes and the scores for each episode was collected as the previous scheduler.

Details of the Implementation

1. **Number of Episodes**: The algorithm is run for a defined number of episodes, in this case, 20.

2. **Initialization**: At the beginning of each episode, the environment is reset to its initial state, and variables to track the score are initialized.

3. **Action Selection**: The Random scheduler selects an action randomly from the available action space using the `env.action_space.sample()` method.
4. **Step Execution**: At each step, the selected random action is executed within the environment using the `env.step` method. The reward is added to the episode's score, and the termination conditions are checked.
5. **Termination**: The loop continues until a termination or truncation condition is met, marking the end of the episode. The final score for the episode is recorded.
6. **Results**: The scores for all episodes are printed, along with the average score across the episodes.

Operation in the Context of the Simulation The Random scheduler is a non-deterministic approach where each action is selected randomly without any specific logic or pattern. In the context of the simulation:

- **Randomness**: By selecting actions randomly, the Random scheduler introduces variability and unpredictability into the simulation.
- **Simplicity**: The implementation is simple and requires no complex logic or knowledge about the characteristics or requirements of the actions.
- **Fairness**: As all actions are equally likely to be selected, the Random scheduler maintains a form of fairness.
- **Limitations**: The Random scheduler does not take into account any specific characteristics or requirements of the actions, such as priority or complexity. This random behavior may lead to suboptimal performance, especially in scenarios where specific action selection logic is needed.

In summary, the Random scheduler provides a basic and unbiased approach to action selection in the simulation. While it lacks sophistication and may lead to less efficient outcomes, it serves as a useful benchmark and introduces an element of randomness that can be valuable in testing and exploration.

2.2.2.3 Q-Learning Model

Agent configuration The agent was designed to implement the Q-learning algorithm in the context of a Wireless Sensor Network (WSN) environment. The following hyperparameters were used:

- **Learning Rate** (α): The learning rate was set to 0.01, controlling the extent to which new Q-values overwrite the old values.
- **Discount Factor** (γ): The discount factor was set to 0.95, determining the agent's consideration for future rewards.

- **Epsilon** (ϵ): A start value of 1.0 was used for epsilon, which controls the exploration vs exploitation trade-off. This value decays linearly over time, with the decay rate being set to $\frac{\text{start\_epsilon}}{\frac{\text{n\_episodes}}{2}}$, and a final epsilon value of 0.1.

Action Selection Policy The agent used an epsilon-greedy policy, which acts randomly with a probability of ϵ, ensuring exploration of the environment, and greedily (choosing the action with the highest estimated reward) with a probability of $1 - \epsilon$, to exploit the gathered knowledge.

Q-Value Update Rule The Q-value update rule implemented in the agent is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a) \right)$$

where s is the current state, a is the current action, s' is the next state, r is the immediate reward, and $\max_{a'} Q(s', a')$ is the maximum Q-value for the next state.

Training Loop The agent was trained over $10,000$ episodes. Each episode begins with the environment being reset, followed by the agent taking actions until termination or truncation. The actions are chosen using the epsilon-greedy policy described above, and the Q-values are updated according to the chosen action, observed reward, and the next state.

Monitoring and Convergence The temporal difference error was recorded during training, allowing for monitoring convergence. The epsilon value was also decayed linearly across episodes, gradually shifting the balance from exploration to exploitation.

2.2.2.4 Deep Q-Network (DQN) Model

The DQN model consists of a neural network used to approximate the Q-function. The model is implemented using PyTorch and includes the following components:

Neural Network Architecture: The neural network architecture is defined as a PyTorch class named `Network`, inheriting from `nn.Module`. The architecture consists of the following layers:

- **Input Layer**: Flattens the observation space of the environment and connects to the first hidden layer.
- **Hidden Layer**: A dense layer with 64 units, followed by a hyperbolic tangent activation function.
- **Output Layer**: A dense layer that outputs a value for each possible action in the environment's action space.

The class also includes methods for acting based on Boltzmann action selection, computing the loss using the smooth L1 loss function, saving and loading the model parameters.

Hyperparameters The following hyperparameters marked in the table 2.1 are used in the DQN model:

Hyperparameter	**Value**
Discount Rate γ	0.99
Batch Size	32
Buffer Size	1×10^6
Minimum Replay Size	1000
Epsilon Start	1.0
Epsilon End	0.1
Epsilon Decay	3×10^6
Target Update Frequency	1000
Learning Rate	5×10^{-5}

Table 2.1: DQN model Hyperparameters

Paths for saving and logging, as well as other constant values for the training process, are also defined.

Training Loop The DQN is trained using the following process:

1. **Initialization**: The replay buffer, reward buffer, episode reward, episode count, online network, target network, and optimizer are initialized.

2. **Replay Buffer Filling**: Initial transitions are collected to fill the minimum replay size.

3. **Main Training Loop**: The model's main training loop is executed with the following key steps:

    (a) Action selection based on epsilon-greedy policy.

    (b) Execution of the action and storage of the transition.

    (c) Gradient descent step using a sampled mini-batch from the replay buffer.

    (d) Target network update based on a frequency parameter.

    (e) Logging and saving at specified intervals.

During training, progress is monitored through logging and TensorBoard, which includes tracking the average reward and the number of episodes. Additionally, the model is saved at regular intervals.

Observations and Future Work The DQN model represents a powerful tool for solving complex problems in reinforcement learning. By training on the specific environment and adjusting hyperparameters, the model can be tailored to various tasks.

2.2.2.5 Stable Baselines 3 (SB3) Model

Before implementing and training the model using the Stable Baselines 3 library, it was essential to ensure that our custom environment adhered to the necessary interface requirements. This guarantees that the environment can correctly interact with the chosen algorithm (in this case, Deep Q-Network) provided by the library. The following steps detail the procedure carried out to validate the environment's compatibility, declare the model, train it, and finally test it to evaluate performance.

Environment Compatibility Check:

To ensure that the custom environment is compatible with the Stable Baselines 3 library, we performed a compatibility check using the following predefined function:

```
check_env(env, warn=True, skip_render_check=True)
```

Model Declaration

The Deep Q-Network (DQN) model was declared with the following specifications:

```
model = DQN("MlpPolicy", env, verbose=1)
```

Where:

- `"MlpPolicy"` refers to the multi-layer perceptron policy.
- `env` is the custom environment used for training.
- `verbose=1` enables detailed logging during the training process.

Training the Model

The model was trained for a total of 200,000 timesteps, with logging information printed every 4 intervals, as follows:

```
model.learn(total_timesteps=200_000, log_interval=4)
```

Testing the Model

After training, the model was tested over 20 episodes, following a loop to step through the environment based on actions predicted by the model. This resulted in an average score that indicates the performance of the model.

In this section, we performed a thorough verification of the environment's compatibility with the Stable Baselines 3 library, ensuring a smooth interface between the custom environment and the pre-defined Deep Q-Network (DQN) algorithm. After confirming compatibility, the DQN model was declared, trained on the environment for a specified number of time steps, and subsequently evaluated over multiple episodes. The process showcased the utility of using established libraries in reinforcement learning and demonstrated how they could be leveraged to apply complex algorithms to custom problems efficiently.

Conclusion

This chapter represented a comprehensive exploration of the design and simulation processes utilized within the Wireless Sensor Network environment. By incorporating various technologies such as Gymnasium, Stable Baselines 3, and additional tools and libraries, the simulation was made robust and versatile.

The distinct design elements of the WSN environment were carefully examined, and the rationale behind the key choices made in constructing the simulation was outlined. These foundational concepts set the stage for the subsequent deployment of various models, algorithms, and schedulers.

The simulation section showcased five different methodologies, including Q-Learning, Deep Q-Network, Round Robin Scheduler, Random Scheduler, and a model leveraging Stable Baselines 3. Each approach was detailed with its unique characteristics and operational logic.

Through the in-depth description of the design and simulation, this chapter laid the groundwork for future investigation and analysis. The next chapter will take these insights further, examining the results of the simulations to understand the strengths and weaknesses of each approach in the context of the WSN environment.

Chapter 3

Results and Discussion

Introduction

This chapter is dedicated to the analysis and discussion of the findings obtained through the implementation of various algorithms and schedulers applied to the Wireless Sensor Network (WSN) environment. A series of models, including the Random Scheduler, Round Robin Scheduler, Q-Learning, and Deep Q-Network (DQN), were rigorously tested and evaluated.

The chapter is structured as follows: Section 3.1 provides a brief recap of the methodologies employed. Section 3.2 details the results and performance of each algorithm followed by A discussion of the results in Section 3.3. The chapter concludes with a summary of key findings in Last Section.

3.1 Methodology Recap

The study employed a customized gym environment tailored to simulate the WSN within a series of models and scheduling algorithms.

Customized Gym Environment The WSN environment was designed within the OpenAI Gym framework, enabling a consistent interface for training and testing various algorithms. The environment encapsulated specific features and constraints of a WSN, such as the topology, connectivity, timesteps, and other attributes that mirror real-world scenarios.

Experimental Setup The experiment involved training and evaluating each model over a predefined number of episodes within the customized gym environment. Key metrics such as reward convergence, latency, and average reward were captured and analyzed. TensorBoard was used for visualizing the training progress of the DQN model.

Random Scheduler The Random Scheduler served as a baseline, selecting actions randomly without any underlying logic or optimization. It provided an essential comparison point for more sophisticated algorithms.

Round Robin Scheduler The Round Robin Scheduler implemented a systematic, cyclical approach where actions were selected in a repeating sequence. This ensured a fair distribution and avoided favoring any particular action.

Q-Learning Model The Q-Learning model utilized a form of model-free reinforcement learning to explore and learn optimal action paths in the environment. Through iterative learning and value updating, the model adapted to maximize cumulative rewards.

Deep Q-Network (DQN) Model The DQN is an extension of traditional Q-Learning using deep neural networks. It enabled handling more complex state spaces and brought increased efficiency and robustness to the learning process.

Stable Baselines 3 (SB3) Model Stable Baselines 3 provided a set of high-quality implementations of reinforcement learning algorithms in Python. For this study, the SB3 library was used to implement the DQN model, utilizing predefined functionalities and best practices within the library. It allowed for rapid development, experimentation, and robust performance within the customized gym environment.

These methodologies laid the foundation for the study and facilitated an in-depth exploration of different strategies within the WSN context.

3.2 Results

This section provides a detailed analysis of the models implemented and evaluated in this study. Various algorithms, including Random Scheduler, Round Robin, Q-learning, Deep Q-Network (DQN), and Stable Baseline 3 Model, are assessed based on their performance, time latency, and other key metrics. A comprehensive review of each algorithm's behavior, strengths, and weaknesses is elucidated to shed light on the nuances of event scheduling and sensor selection.

3.2.1 Random Scheduler

The Random Scheduler served as a baseline scheduler for comparison with more advanced algorithms. The results were obtained by executing the scheduler for 20 episodes, and the average reward score result was 285.047.
A sample output for an episode is provided as follows:

```
Episode:1 Score:291.84 {'captured': 717, 'non-captured': 10,
'sensor 0': {(1, 994), (1, 1000), (1, 996), (1, 993), (1, 999)},
'sensor 1': set(), 'sensor 2': set(), 'sensor 3': {(1, 1000),
(1, 999), (1, 994)}, 'sensor 4': set(), 'sensor 5': {(1, 996),
(1, 993), (1, 1000)}, 'sensor 6': set(), 'sensor 7': set(),
'sensor 8': {(1, 1000)}, 'sensor 9': {(1, 996), (1, 1000)},
'sensor 10': set(), 'sensor 11': {(1, 999)}}
```

The dictionary output provides detailed information about the number of captured events, the non-captured events, and the buffer status for each sensor. It's important to note that the number of non-captured events refers to the events that were generated but not captured by the sensors. Consequently, these events were never added to the sensor buffers. The details of the buffer status for each sensor further assist in understanding the efficiency and performance of the Random Scheduler.

The time latency in timesteps was plotted, and the average, maximum, and minimum latencies for an episode were computed and analyzed, providing insights into the performance of the Random Scheduler in terms of time efficiency.

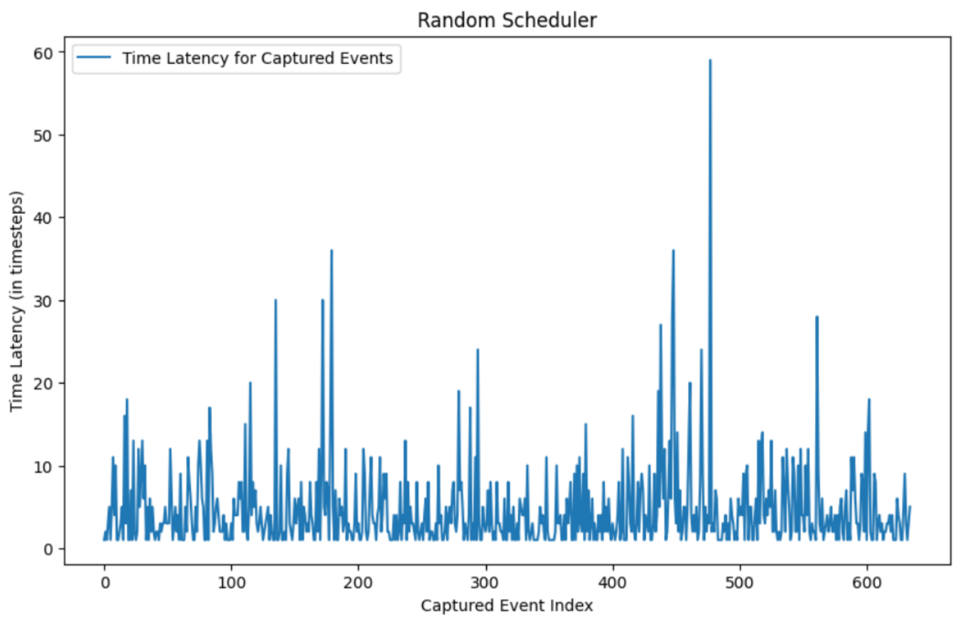

Figure 3.1: Time latency for the Random Scheduler.

The printed output for the time latency analysis from the figure 3.1 is given as:

```
Average Time latency: 4.67, min: 1.00, max: 59.00
```

This data further assists in understanding the behavior of the Random Scheduler with respect to the latency of capturing events.

3.2.2 Round Robin Scheduler

The Round Robin scheduler was next examined to observe its efficacy in comparison to the random scheduler. In this approach, the scheduling is done in a systematic and periodic manner, looping through the sensors to ensure a fair allocation of events.

The experimental results for the Round Robin scheduler over 20 episodes are presented below. A notable improvement was observed in both the reward and time latency when compared to the random scheduler.

- **Average Reward**: 308.34
- **Time Latency**: Average: 3.62, Minimum: 1.00, Maximum: 21.00

Figure 3.2 illustrates the time latency plot for the Round Robin scheduler. A representative episode printout is also provided:

```
Episode:1  Score:311.46  {'captured': 688, 'non-captured': 9, 'sensor 0':
set(), 'sensor 1': set(), 'sensor 2': set(), 'sensor 3': set(), 'sensor 4':
set(), 'sensor 5': {(1, 990), (1, 997), (1, 1000)}, 'sensor 6': set(),
'sensor 7': set(), 'sensor 8': {(1, 997), (1, 1000)}, 'sensor 9': {(1, 990),
(1, 997), (1, 1000)}, 'sensor 10': set(), 'sensor 11': set()}
```

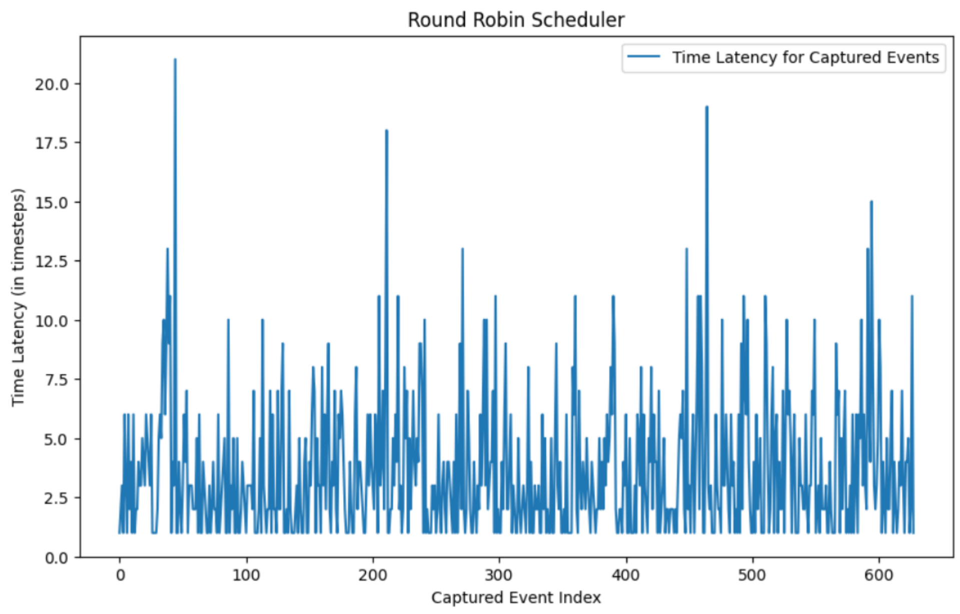

Figure 3.2: Time Latency for Round Robin Scheduler

These results highlight the more effective handling of event scheduling in the Round Robin approach, leading to an enhancement in the performance metrics.

3.2.3 Q-learning Model

Performance Analysis The Q-learning model was tested for performance, and the results were indicative of certain limitations in handling the complexity of the

Wireless Sensor Network environment. The average score reward for 20 episodes was measured at 214.78, a value that, although promising, reveals certain challenges in optimizing the scheduling patterns.

Time Latency Analysis Figure 3.3 illustrates the time latency plot for the Q-learning scheduler.

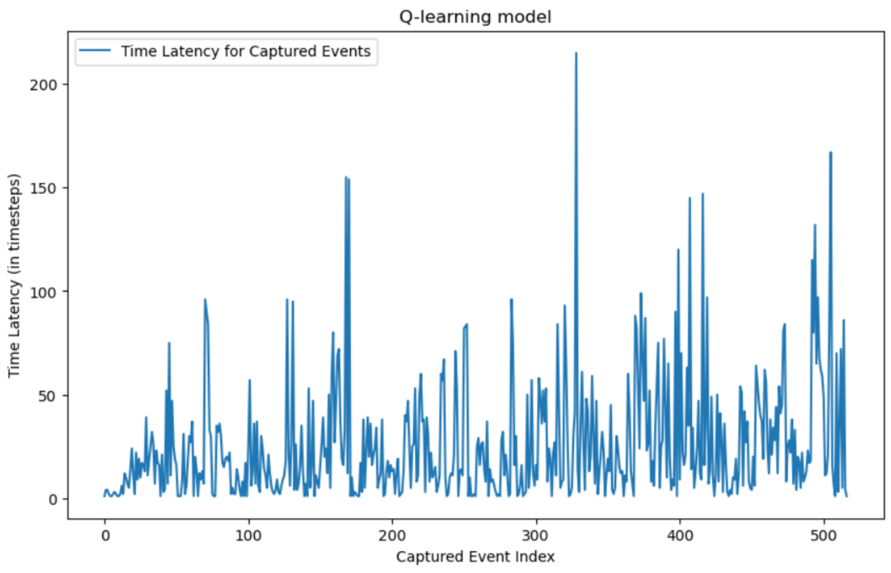

Figure 3.3: Time Latency for Q-learning model

The model performed poorly comparing to the previous schedulers from the average time latency and the maximum as well as the average reward which gave the proposition for further refinements and the integration of Neural Networks.
The printed output for the time latency in timesteps is given as:

```
Average Time latency: 25.54, min: 1.00, max: 215.00
```

- **Exploration vs Exploitation**: The epsilon-greedy policy, although designed to balance exploration and exploitation, may not have provided enough adaptability for the model to learn optimal policies in the given scenario.

- **Simplicity of Q-Learning**: The tabular approach of Q-learning, while elegant, can be limited in capturing the high-dimensional state space of the WSN environment.

- **Convergence Challenges**: The chosen hyperparameters and the Q-value update rule may have led to slow convergence, impacting the overall efficacy of the model.

Integration with Neural Networks: Towards DQN: The performance of the Q-learning model offers valuable insights into the direction of improving the scheduling algorithm. Specifically, the combination of Q-learning with a neural network leads to the DQN model, an architecture that can potentially overcome some of the aforementioned challenges.

3.2.4 Deep Q-Network (DQN) Model

The DQN model represents a substantial part of this study, and its results are presented in this section. Training for 170 episodes with 1000 timesteps each, the model's performance was carefully analyzed.

Reward Function Progress The evolution of the reward function was monitored throughout the training. Figure 3.4 depicts this progression, showing fluctuations before converging to an approximate value of 290. These fluctuations signify the exploration phase, which eventually stabilizes after nearly 120 episodes.

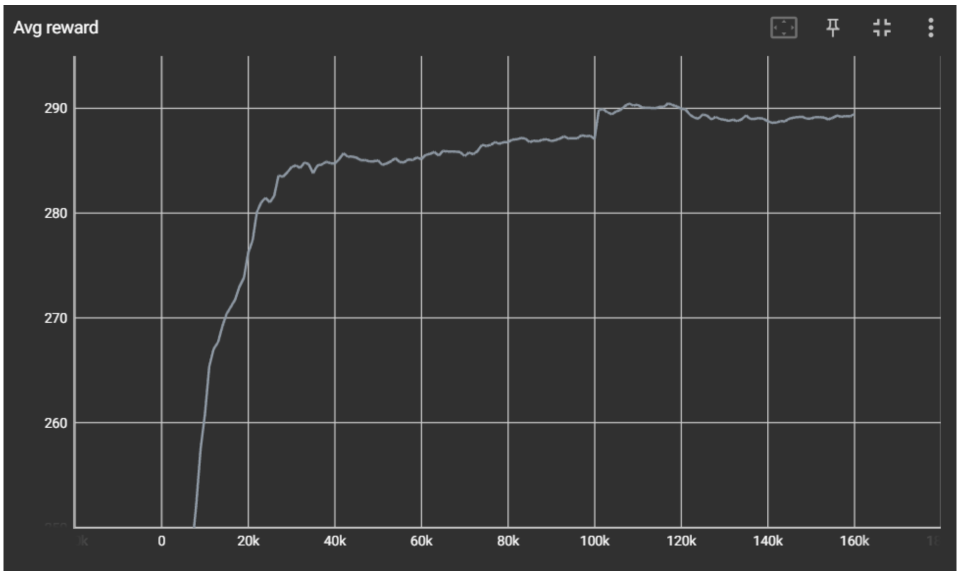

Figure 3.4: Progress of Reward Function in DQN Model

Episode vs. Timestep Analysis In Figure 3.5, the number of episodes is plotted against the timesteps, representing a typically affine line.

Time Latency Analysis The time latency results for the DQN model are as follows:

- Average Time Latency: 4.85
- Minimum: 1.00

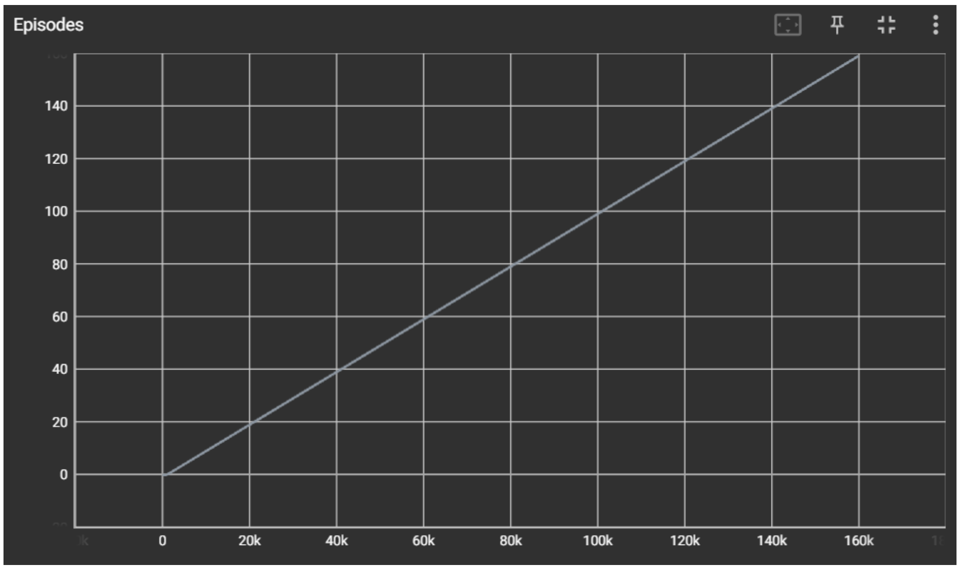

Figure 3.5: Number of Episodes vs. Timesteps in DQN Model

- Maximum: 44.00

From these results, we can observe that the DQN model performed better than the Random Scheduler in terms of the maximum latency. However, the Round Robin Scheduler still maintains a lead with a lower average and maximum time latency. This comparison illustrates the trade-offs and relative benefits of different scheduling techniques. Figure 3.6 provides a graphical representation of the time latency for the DQN model.

Episode and Sensor Selection Printout The sensor positions and selection frequency are represented in an array:

```
array([[7.3199391e-01, 5.9865850e-01, 7.1000000e+01],
       [1.5601864e-01, 1.5599452e-01, 5.7000000e+01],
       [5.8083612e-02, 8.6617613e-01, 4.8000000e+01],
       [6.0111499e-01, 7.0807260e-01, 6.5000000e+01],
       [2.0584494e-02, 9.6990985e-01, 3.9000000e+01],
       [8.3244264e-01, 2.1233912e-01, 7.1000000e+01],
       [1.8182497e-01, 1.8340451e-01, 4.0000000e+01],
       [3.0424225e-01, 5.2475643e-01, 7.0000000e+01],
       [4.3194503e-01, 2.9122913e-01, 4.9000000e+01],
       [6.1185288e-01, 1.3949387e-01, 4.3000000e+01],
       [2.9214466e-01, 3.6636186e-01, 8.3000000e+01],
       [4.5606998e-01, 7.8517598e-01, 7.2000000e+01]], dtype=float32)
```

This data implies that the agent learned to select certain sensors more frequently depending on their location.

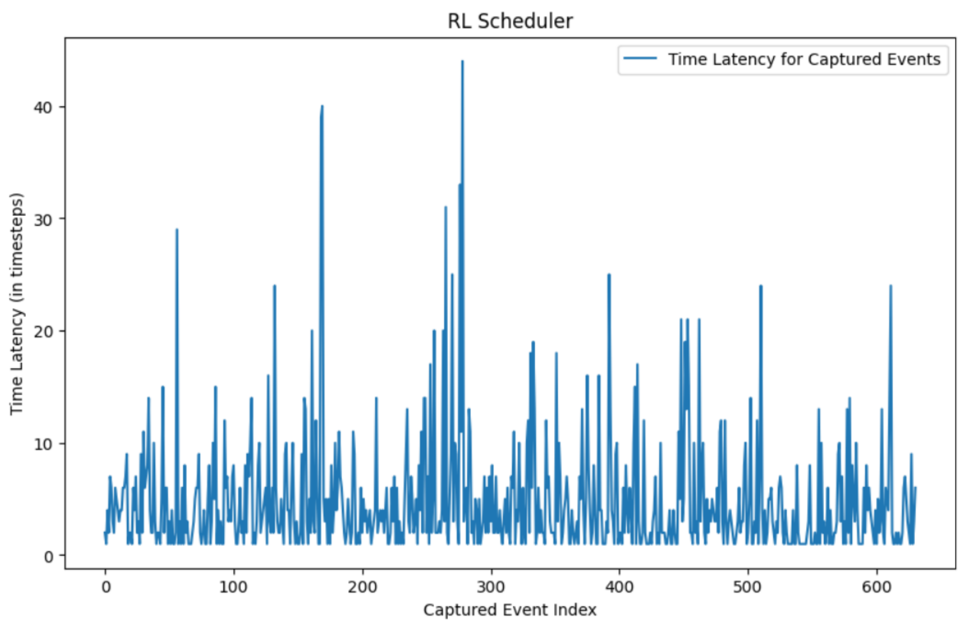

Figure 3.6: Time Latency for the trained agent

Conclusion The DQN model's results showcase its capability to learn sophisticated scheduling patterns, leading to a harmonious balance between exploration and exploitation. The convergence of the reward function after approximately 120 episodes reflects the model's learning progression, while the differential selection of sensors emphasizes the model's adaptability. Despite the successful performance in the context of sensor selection, it is noteworthy that the DQN model exhibited time latency metrics that were superior to the Random Scheduler but not as efficient as the Round Robin Scheduler. This highlights a potential area for future improvement. Overall, the DQN's approach, visualized in the TensorBoard figures and time latency plot, underscores its potential in handling event scheduling, laying the groundwork for further enhancements and applications in the field of wireless sensor networks.

3.2.5 Stable Baseline 3 Model

The SB3 model was employed to explore and understand the complex task of sensor selection and event capturing in the given context. As a more advanced and refined model, SB3 was anticipated to demonstrate an intelligent balance in selecting sensors and optimizing rewards. The following subsections detail the training process, testing outcomes, and specific behavioral patterns observed during the simulation, providing comprehensive insights into the model's performance and potential areas for improvement.

Training and Testing: The Stable Baselines 3 model was trained using the command:

```
model.learn(total_timesteps=100_000, log_interval=4)
```

After training, the model was tested for 20 episodes, yielding an average score reward of 305.27. The model demonstrated certain patterns in selecting sensors and captured events, though some noticeable aspects require attention.

Episode Analysis One episode of testing revealed the following details:

- Score: 317.66
- Captured events: 651
- Non-captured events: 43
- Sensor Selection: Varied, with some sensors experiencing full buffers.

The corresponding sensor tensor indicated limited exploration by the model, focusing on certain sensors while neglecting others. This behavior, while contributing to rewards, may be considered a concern as shown in the sensor tensor (third column):

```
array([[7.3199391e-01, 5.9865850e-01, 2.1000000e+01],
       [1.5601864e-01, 1.5599452e-01, 1.0000000e+00],
       [5.8083612e-02, 8.6617613e-01, 0.0000000e+00],
       [6.0111499e-01, 7.0807260e-01, 2.0400000e+02],
       [2.0584494e-02, 9.6990985e-01, 0.0000000e+00],
       [8.3244264e-01, 2.1233912e-01, 0.0000000e+00],
       [1.8182497e-01, 1.8340451e-01, 0.0000000e+00],
       [3.0424225e-01, 5.2475643e-01, 1.2500000e+02],
       [4.3194503e-01, 2.9122913e-01, 0.0000000e+00],
       [6.1185288e-01, 1.3949387e-01, 1.4300000e+02],
       [2.9214466e-01, 3.6636186e-01, 2.0700000e+02],
       [4.5606998e-01, 7.8517598e-01, 2.2000000e+01]], dtype=float32)
```

The plot in Figure 3.7 illustrates that most events have time latency between 1 and 20, with two distinct clusters around 90 and between 100 to 130.

Time Latency Analysis The time latency statistics for the captured events were:

- Average Time Latency: 7.49
- Minimum: 1.00
- Maximum: 133.00

Comparatively, the Stable Baselines 3 model exhibited a higher average time latency than previous models, such as the Random Scheduler (4.67), Round Robin (3.62), and the DQN model (4.85). The issue of long latency and high number of non-captured events indicates potential areas for further refinement.

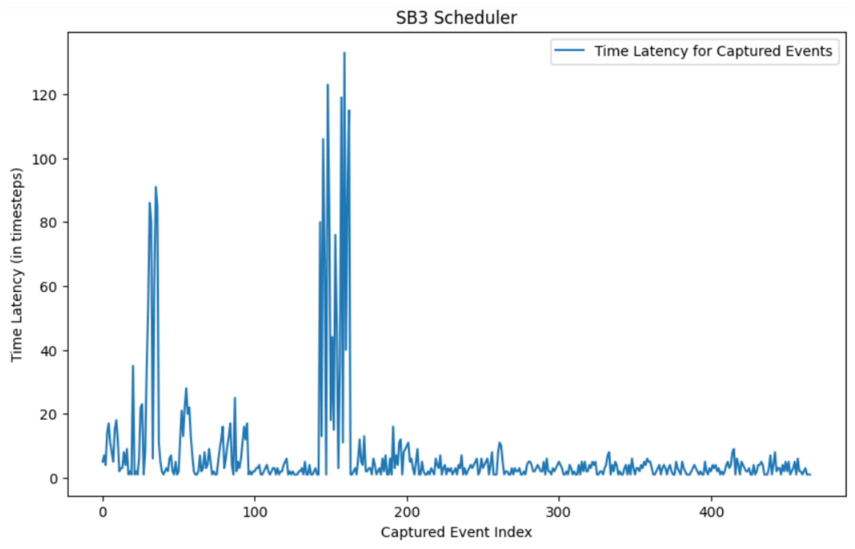

Figure 3.7: Time Latency for the Stable Baselines 3 Model.

While the model succeeded in earning rewards, its behavior, notably in sensor selection and lack of exploration, warrants attention. These results underscore the complexity of the task and offer insights for future work, especially in balancing exploration, exploitation, and efficient event capturing.

3.3 Discussion

The evaluation of different scheduling algorithms for Wireless Sensor Network event scheduling reveals insightful results. In this section, we focus on the comparison and discussion of the implemented models, emphasizing the strengths and weaknesses of the DQN model relative to the other approaches.

Random Scheduler The Random Scheduler provided a baseline approach but lacked the ability to learn and adapt to the environment. Its performance was erratic and often suboptimal, making it an insufficient solution for complex scheduling tasks.

Round Robin Scheduler The Round Robin Scheduler offered a more systematic approach but still struggled with flexibility and efficiency. While it performed better than the random approach, it could not fully adapt to the dynamic nature of WSN scheduling.

Q-Learning Model The Q-learning model introduced learning capabilities but faced challenges in scalability and convergence. Though promising, it required further enhancements to effectively tackle the high-dimensional state space of WSN.

Deep Q-Network (DQN) Model The DQN model stood out as the most promising solution among the tested algorithms. Combining the principles of Q-learning with the power of neural networks, the DQN model overcame many limitations of the previous approaches:

- **Adaptability**: Unlike the Random and Round Robin Schedulers, the DQN model was capable of learning complex scheduling patterns, adapting to changes in the environment.
- **Efficiency**: In contrast to the tabular Q-learning approach, the DQN model leveraged Neural Networks to capture high-dimensional relationships, resulting in more efficient learning and better performance.
- **Balance between Exploration and Exploitation**: The DQN model demonstrated a nuanced balance in exploring new possibilities and exploiting known knowledge, outperforming other models in the trade-off.
- **Potential for Further Enhancement**: The DQN model provides a robust foundation, with opportunities for further tuning and integration with other methods to achieve even better results.

Stable Baseline 3 Model This model builds upon the principles of DQN, adding further optimizations. Although facing some issues with sensor selection and non-captured events, it indicated the potential for fine-tuning and adaptation.

Comparison of DQN, Round Robin, and Random Schedulers The following table resumes the different values and results for the different algorithms tested:

Algorithm	Avg Reward	Avg Time Latency	Min Latency	Max Latency
Random Scheduler	285.047	4.67	1.00	59.00
Round Robin	308.34	3.62	1.00	21.00
Q-learning	214.78	25.54	1.00	215.00
DQN	295.28	4.85	1.00	44.00
SB3	305.27	7.49	1.00	133.00

Table 3.1: Performance Metrics of Different Scheduling Algorithms

Based on the table 3.1, the Round Robin scheduler appears to be the most effective, with the highest average reward of 308.34 and the lowest average time latency of 3.62. DQN also performs well with an average reward of 295.28, closely followed by SB3 at 305.27, but with slightly higher time latencies. Q-learning has the least effective performance, indicated by its lowest average reward of 214.78 and the highest average time latency of 25.54. The Random scheduler offers a balanced performance but falls short of Round Robin and DQN in terms of both average reward and time latency.

As detailed in the Table, the different scheduling algorithms exhibit varying levels of performance in terms of average rewards and time latency. Particularly noteworthy is the time latency, which directly impacts the responsiveness and efficiency of the wake-up radio systems. While the tabulated metrics provide a quantitative overview, it is often beneficial to visualize these differences to grasp the underlying behavior more intuitively.
To that end, Figure 3.8 presents a plot comparing the time latency of three selected algorithms—Random Scheduler, Round Robin, and DQN. This plot captures only the first 100 values for each algorithm to offer better visibility and focus on the initial behavior.

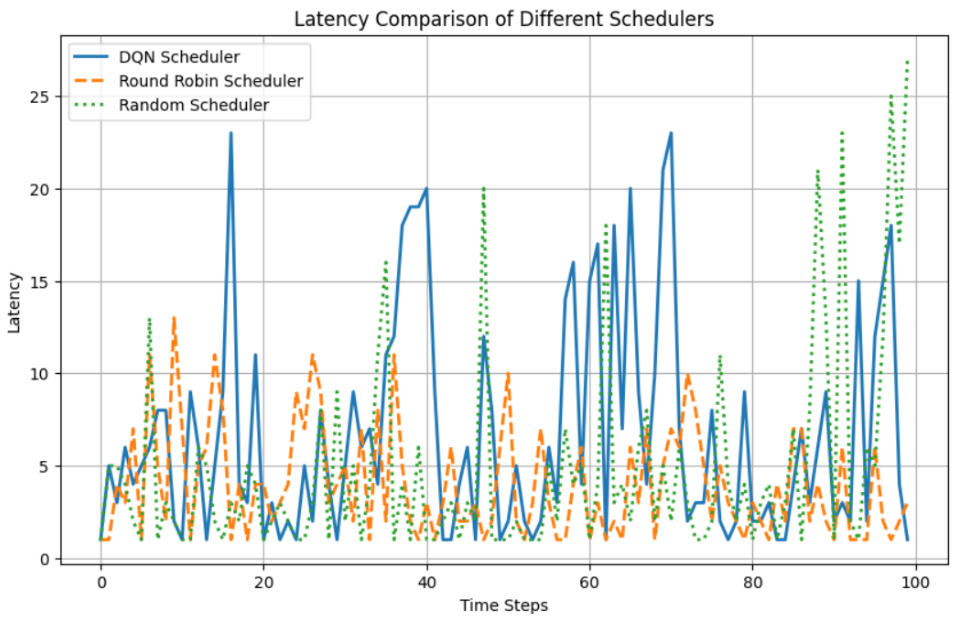

Figure 3.8: Comparison of Time Latency.

The time latency plot shown corroborates the data summarized in the latter Table. As evidenced by the plot, the Round Robin scheduler consistently demonstrates the lowest time latency among the three algorithms. Its performance is noticeably superior, making it still more efficient in terms of time latency for our wake-up radio systems. Following the Round Robin, the DQN scheduler also shows commendable performance, but it's slightly inferior compared to the Round Robin scheduler. Despite that it presents some pics, it can be considered as a viable alternative for applications where slight variability in latency can be tolerated. The Random scheduler lags behind the other two, confirming its position as the least effective in terms of time latency. Its sporadic peaks in latency could potentially introduce inefficiencies into the system, making it a less desirable option for real-time or time-sensitive applications. Thus, the plot successfully validates the tabulated findings, offering a more granular look into the time-latency behavior of each algorithm.

For further comparison, Figure 3.9 presents the average rewards per episode for the three considered schedulers: DQN, Round Robin, and Random. The plot illustrates the distinct behavior and performance of each model across the episodes.

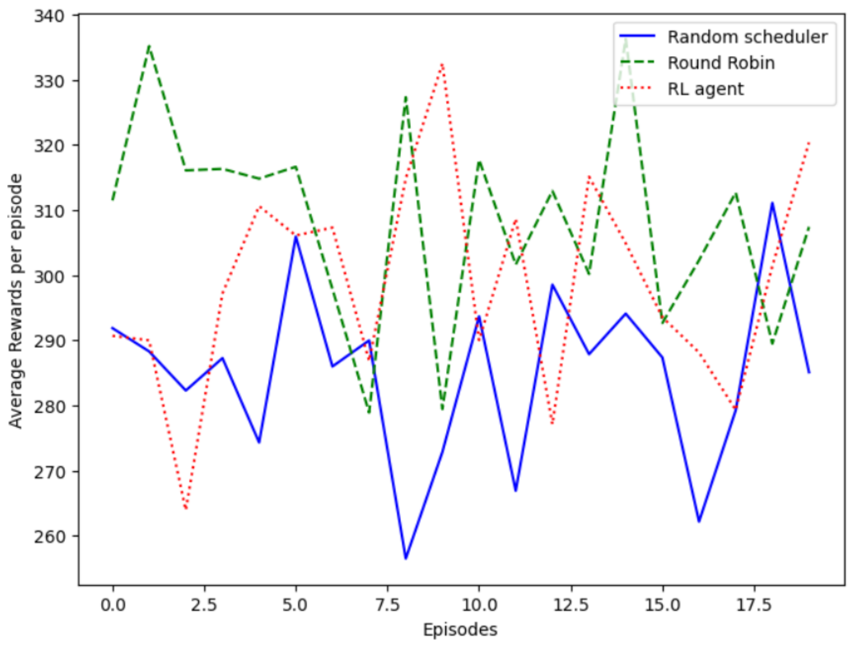

Figure 3.9: Comparison of average rewards per episode.

- **Round Robin Scheduler**: This model consistently outperformed the Random Scheduler, demonstrating stability and higher average rewards. Its systematic approach to scheduling led to a more reliable performance.
- **Random Scheduler**: The Random Scheduler's performance was the most erratic, frequently falling below the Round Robin model. Its lack of learning and adaptability rendered it suboptimal in the majority of episodes.
- **DQN Model**: The DQN model exhibited a more complex behavior, bouncing between the other two models. While it occasionally outperformed the Round Robin model, it sometimes fell below the Random Scheduler, and most of the time, it was positioned between them. This fluctuating performance highlights the DQN's learning process, adaptability, and sensitivity to the environment's dynamics. The model's ability to sometimes outperform the Round Robin model showcases its potential, although the inconsistencies indicate room for further tuning and improvement.

Boxplot Figure 3.10 provides a boxplot representation comparing the performance of the three different scheduling algorithms. The boxplot visualizes the distribution of average rewards per episode for each algorithm, capturing key statistics like median, quartiles, and outliers.

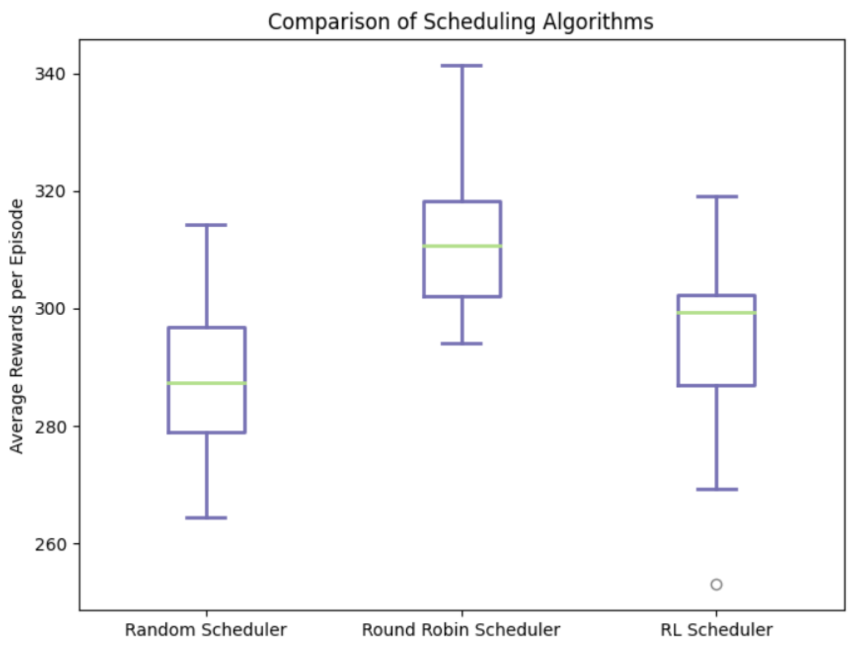

Figure 3.10: Boxplot comparison of average rewards per episode.

Comparison of Medians: The Round Robin scheduler boasts the highest median value, indicating its superior overall performance. The DQN (RL) scheduler closely follows, while the Random scheduler displays the lowest median, pointing to less effective scheduling decisions.
Spread of Rewards: The Round Robin scheduler has a wider range above its third quartile compared to below its first quartile, showing a propensity for higher rewards. The DQN (RL) scheduler's spread is more balanced, while the Random scheduler has the lowest bounds.
Outliers and Whiskers: The DQN (RL) scheduler shows one outlier below its box, which is the lowest value across all algorithms. It indicates a possible scenario where the RL-based scheduling performs poorly. However, its whiskers are balanced, suggesting a uniform distribution of rewards around the median. Round Robin, with its asymmetrical whiskers, shows that it's more likely to achieve higher rewards. The Random scheduler has nearly symmetrical whiskers, revealing a balanced yet limited spread of rewards.
Skewness: The Round Robin scheduler's data marks a right-skewed distribution, indicated by a longer upper whisker and a median that is central, which in this case means a tendency towards higher rewards. The DQN (RL) scheduler shows a left skewness towards higher rewards, with its median closer to its Q3. The Random scheduler's data is slightly right skewed towards lower rewards, with its median closer

to its Q1.

Overall, the Round Robin scheduler appears to be the most promising with the highest propensity for higher rewards, despite its greater variability. The DQN (RL) scheduler offers a reliable but slightly less rewarding alternative, its performance occasionally dipping as indicated by a single outlier. The Random scheduler performs the least effectively, confirmed by its lowest median and a more confined range of rewards.

The plot and observations reinforce the conclusion that while the DQN model offers a promising approach with learning capabilities, it requires further refinement to consistently outperform simpler models like the Round Robin Scheduler. The complex and nuanced behavior of the DQN model presents exciting opportunities for exploration and enhancement in future work.

Conclusion

The results and subsequent discussion have shed light on the complexity and challenges involved in the task of event scheduling. By exploring various algorithms, this chapter has uncovered key insights and nuances that will be vital in the ongoing pursuit of optimal solutions. The DQN model emerges as a robust and versatile solution for WSN event scheduling. It was built upon the foundational principles of Q-learning to offer a scalable, adaptive, and efficient approach. The exploration of SB3 hints at the future direction of research, emphasizing continuous innovation and refinement in the pursuit of optimal scheduling solutions. In summary the analysis presented here sets the stage for future work, opening new avenues for research and development in the realm of scheduling algorithms.

Conclusion

The journey of this project has been a deep dive into the multidimensional problem of energy consumption in IoT systems. The focus was on creating a robust framework that amalgamates the power of Wake-Up Radio Systems, Intelligent Scheduling, Machine Learning, and particularly Reinforcement Learning through the use of Deep Q-Networks. The framework's validation process involved several key steps, implemented to meet high standards of rigor and reproducibility. Initially, we constructed a customized simulation environment tailored to our specific needs using the Gym API, a well-established standard for reinforcement learning environments. This custom environment served as the testing ground for our experimental models. Following this, we validated the environment's functionality and reliability using Stable Baselines3, a renowned library for reinforcement learning. Importantly, the study didn't just rely on a custom-built DQN model, it also employed the predefined DQN model available in Stable Baselines3. This served as a baseline model against which our custom-built DQN model was compared, allowing for a more nuanced understanding of its performance and potential advantages.

The empirical results of this study are twofold. On one hand, they establish the marked improvement in energy efficiency that the integration of DQN and Wake-Up Radio Systems brings to the table. This is not merely an academic exercise but offers a substantial contribution to the field of IoT, particularly in industrial and home automation where the optimization of energy resources is a critical concern. On the other hand, the study extends beyond mere numbers to offer methodological advancements. It underscores the applicability of intelligent scheduling algorithms, shedding light on how Q-Learning mechanisms can be adapted for solving intricate, real world problems.

The theoretical scaffolding of this work contributes to the extant literature that calls for the application of Machine Learning algorithms in IoT scenarios for better energy management and scheduling. Moreover, the model's flexibility to adapt to various scenarios and environmental conditions promises a wider applicability, not restricted to just the focus of this study.

Looking ahead, several avenues open up for future research. This includes but is not limited to, exploring different algorithmic strategies, focusing on real-world implementations for empirical validations, and extending the model to adapt to various environmental conditions and different types of IoT systems. Such future works can build on the robustness and scalability demonstrated by the current study, providing a fertile ground for both theoretical and practical advancements in the field.

In closing, the framework developed in this project doesn't merely hint at its future potential but lays down a strong foundational path for forthcoming explorations. The road ahead is long and promises to be exciting, teeming with opportunities for both academic and practical implementations.

Bibliography

[1] Andrea Zanella, Anay Ajit Deshpande, and Federico Chiariotti. "Low-latency massive access with multicast wake-up radio". In: *Department of Information Engineering, University of Padova, Via G. Gradenigo 6/B, Padova, Italy* ().

[2] Junya Shiraishi, Anders E Kalør, Federico Chiariotti, Israel Leyva-Mayorga, Petar Popovski, and Hiroyuki Yomo. "Query timing analysis for content-based wake-up realizing informative IoT data collection". In: *IEEE Wireless Communications Letters* 12.2 (2022), pp. 327–331.

[3] Olga Vikhrova, Federico Chiariotti, Beatriz Soret, Giuseppe Araniti, Antonella Molinaro, and Petar Popovski. "Age of information in multi-hop networks with priorities". In: *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE. 2020, pp. 1–6.

[4] Baran Tan Bacinoglu, Elif Tugce Ceran, and Elif Uysal-Biyikoglu. "Age of information under energy replenishment constraints". In: *2015 Information Theory and Applications Workshop (ITA)*. IEEE. 2015, pp. 25–31.

[5] Igor Kadota, Abhishek Sinha, Elif Uysal-Biyikoglu, Rahul Singh, and Eytan Modiano. "Scheduling policies for minimizing age of information in broadcast wireless networks". In: *IEEE/ACM Transactions on Networking* 26.6 (2018), pp. 2637–2650.

[6] Josefine Holm, Anders E Kalør, Federico Chiariotti, Beatriz Soret, Søren K Jensen, Torben B Pedersen, and Petar Popovski. "Freshness on demand: Optimizing age of information for the query process". In: *ICC 2021-IEEE International Conference on Communications*. IEEE. 2021, pp. 1–6.

[7] Federico Chiariotti, Anders E Kalør, Josefine Holm, Beatriz Soret, and Petar Popovski. "Scheduling of sensor transmissions based on value of information for summary statistics". In: *IEEE Networking Letters* 4.2 (2022), pp. 92–96.

[8] Anamika Sharma and Siddhartha Chauhan. "A distributed reinforcement learning based sensor node scheduling algorithm for coverage and connectivity maintenance in wireless sensor network". In: *Wireless Networks* 26.6 (2020), pp. 4411–4429.

[9] Anis Elgabli, Hamza Khan, Mounssif Krouka, and Mehdi Bennis. “Reinforcement learning based scheduling algorithm for optimizing age of information in ultra reliable low latency networks”. In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2019, pp. 1–6.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. URL: `https://gym.openai.com/`.

[11] Antonin Raffin, Ashley Hill, Adam Gleave, Maximilian Ernestus, and Anssi Kanervisto. *Stable Baselines3: Reliable Reinforcement Learning Implementations*. 2020. URL: `https://github.com/DLR-RM/stable-baselines3`.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. URL: `https://pytorch.org/`.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. ISBN: 9780262039246.