Journal of Information Systems Engineering and Management
2025, 10(24s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Intelligent Scheduling in Wireless Sensor Networks using Reinforcement Learning based Deep Q-Networks

Haripriya R [1], Suresh M[2], Vinutha CB[3]
[1,2] *Dept.of Electronics and Communication Engineering, SSIT, Sri Siddhartha Academy of Higher Education, Tumakuru-572105, Karnataka, India*
[3]*Dept.of Electronics and Communication Engineering, Presidency University, Bengaluru, Karnataka, India*

ARTICLEINFO

Received: 25 Dec 2024

Revised: 12 Feb 2025

Accepted: 24 Feb 2025

ABSTRACT

The basic challenge in spatially distributed resource restricted Wireless Senor Network (WSN) is to prolong the network lifetime maintaining its performance criterions like energy efficiency, coverage and connectivity. To achieve this, scheduling can be performed in sensor nodes in terms of their wake up time periods. There are many traditional scheduling mechanisms developed. But, sensor node scheduling still poses a challenge when the WSN becomes dynamic in nature. In this paper, Deep Q-network modeling is carried out with Reinforcement scheduler. The advancement in reinforcement learning and particularly Deep Q-Network (DQN) algorithms provides an efficient tool for implementing intelligent scheduling. When an agent is trained using these techniques, its ability to differentiate between events improves thus, optimizing the scheduling process. The presented scheduling algorithm is evaluated and compared with traditional scheduling algorithms for the same network characteristics. The DQN modeling shows promising potential to intelligently schedule the sensor nodes in terms of energy consumption, average rewards and also minimum and maximum latencies.

Keywords: Wireless Sensor Networks, Machine Learning Algorithms, Scheduling Technique, Deep Q-Networks

INTRODUCTION

Wireless Sensor networks (WSN) constitutes a huge number of small sensor nodes which aggregate physical data from real time environment [1]. The data gathered may be used for many applications in IOT, medical field, wild life surveillance, military, urban communication and many more. The sensed data is transmitted and received simultaneously through wireless link [2]. The main constraint in WSN is its limited energy [3], computation and storage resources. Because of the increase in wireless link for various applications, the scale of WSNs is greatly extended. Since there exists high density communications in WSN, a demand for new methods to meet rigid energy and spectrum requirements also arise [4]. To cater to the main challenges like energy consumption, coverage and lifetime improvement, one important strategy that can be followed is intelligent scheduling using Machine Learning Algorithms. Many researchers have discussed about the probabilistic scheduling techniques and few of them have concentrated on its intelligence as well. Scheduling can be performed with respect to sensor node transmission time, sleeping time, wake up time, data aggregation and so on [1]. This work mainly concentrates on scheduling the wakeup times of sensor nodes. Intelligent Scheduling in WSN is a major technique to that can be followed to handle dynamic nature in WSN. Wake up radio is a technology that allows sensor nodes to respond to requests that can be either ID based or content based [5].

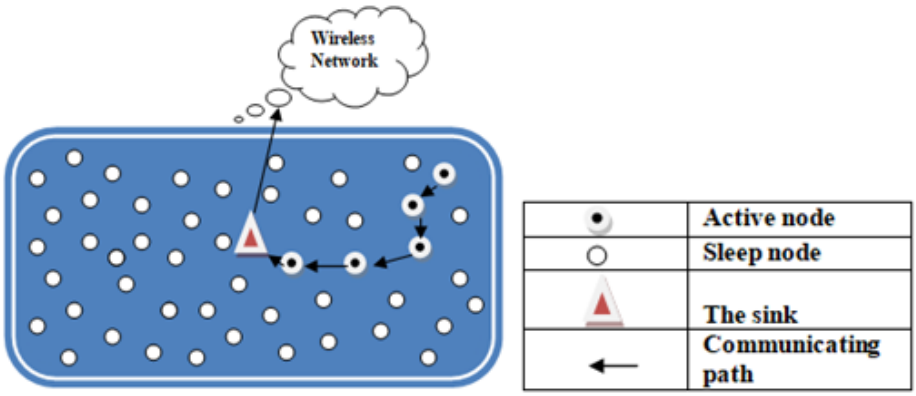

Figure 1: Wireless Sensor Network Structure

A novel scheduling algorithm is proposed which mainly focuses on optimal energy consumption and increasing the state estimation probability in WSN Environment. The work mainly implements reinforcement learning particularly the Deep Q-Network (DQN) algorithm using [6] Stable Baselines3 library to find optimal solution for challenges in dynamic WSN environment. A major challenge to be addressed is the energy consumption and thus is the main objective of the work presented. Out of total number of sensor nodes, those which get intimated from the agent about the event will only be in the active mode [7]. All other nodes remain in sleep mode, so it is obvious that energy expenditure will be reduced [1]. To identify the event occurrence an agent is trained rigorously using Reinforcement learning and then evaluated further.

Environment Description: WSNEnvironment: A Custom Reinforcement Learning Environment for WSN using Gymnasium to simulate the dynamics in the considered network is chosen. The environment plays a crucial role in training and evaluating the scheduling algorithms, especially in reinforcement learning contexts. The design of observation and action spaces is fundamental in reinforcement learning as it dictates what the agent perceives and the actions it can take. In WSNEnvironment, the observation space is a matrix representing the 2D positions of sensors in the network. It tracks the frequency of sensor selection during each episode, enabling the agent to evolve from random behavior to a more strategic approach based on the DQN model. The action space corresponds to the number of sensors in the network, with each action representing the selection of a specific sensor to retrieve data from its buffer.

Environment Dynamics: The dynamic nature of WSNEnvironment [8] allows for realistic simulation of sensor network operations over time: The environment supports customizable sensor deployment, where the number of sensors (`num_sensors`) and their spatial distribution are generated using a Poisson point distribution to ensure randomness and diversity. Sensors operate within defined coverage areas, and the environment simulates the efficient use of sensor energy while maintaining optimal area coverage. Events are generated probabilistically based on a threshold probability (`threshold_prob`). When an event occurs, its location is randomly assigned within the monitored area. Sensors within the event coverage zone detect and log the event, simulating real-world event detection, and store it in their buffers.

Agent Interaction: The 'step' function simulates the agent interaction with the environment, encompassing sensor selection and event capture across multiple time steps as shown in figure 2. The `reset` function reinitializes the environment at the start of each episode. The reward mechanism is centered on minimizing the Age of Information (AoI), a critical factor for optimizing state estimation. The agent is rewarded based on its effectiveness in reducing AoI, encouraging strategic sensor selection. `WSNEnvironment` offers a robust and flexible platform for experimenting with and developing advanced scheduling strategies, particularly in the context of reinforcement learning.

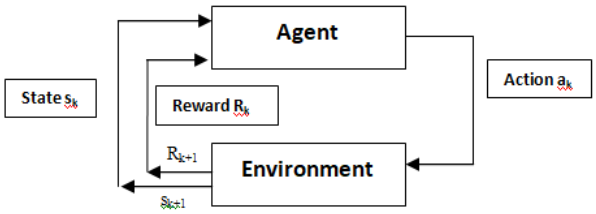

Figure 2 : Agent Environment Interaction [1]

In this study, sensor nodes are strategically placed to ensure coverage and network connectivity within a specified region. It is shown in Figure 1 that the sensor nodes are randomly distributed throughout the remote area. Initially, the nodes possess uniform sensing and communication capabilities. However, the region also contains obstacles that can interfere with these capabilities. Nodes located closer to the base station directly communicate with it.

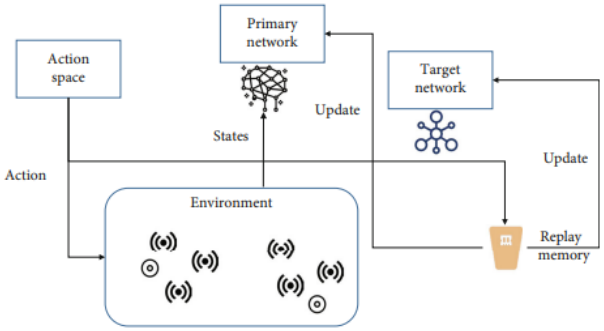

Figure 3 : DQN based scheduling strategy and interacting process between agent and network

Reinforcement Learning is an important technique in Artificial Intelligence (AI), for designing an energy optimizing protocol with an intention to prolong the network lifetime [4]. It is a sub-area of machine learning related to maximization of long term rewards according to the actions taken by particular agent. An agent explores its environment by selecting a specific action at each step and receives a corresponding reward from the environment. The agent has to learn from its experience to identify which is the best action. A sequence of various actions is executed in order to identify the best behavior from the obtained corresponding rewards as shown in Figure 3 [9]. Q-Learning based on RL is one of the most popular and powerful algorithm. It does not need any prior idea about the environment to be structured and its actions depend on Q function. The Q function shows the quality of a particular action at an agent's state. The Q values are updated as

$$Q(st+1, at) = Q(st, at) + \lambda[rt+1 + \psi \max a\, Q(st+1, a) - Q(st, at)] \quad (1)$$

At a time step k, the agent observes environment and identifies its current state s_k. It then takes action a_k based on its observations. The action a_k modifies the environment into a new state s_{k+1}. Further, the agent receives a reward R_{k+1} from the environment [1]. The objective of the agent is to optimize total rewards over the long run. The main task of RL algorithm is to find an optimal policy that achieves maximum rewards.

$$Q_*(s,a) = \max_{\Pi} Q_\pi(s,a) \quad (2)$$

Where, Q(s,a)- value of action a in states.

Author contributions:

A custom WSN environment is constructed using Gymnasium library. This environment is a testing ground to simulate realistic sensor deployment, event detection and energy optimization scenarios. Reinforcement learning is employed to make the model adaptive and intelligent scheduling within the WSN. The agent learns to become optimal by interacting with the environment and by receiving rewards. An optimal action value function is found out by employing Deep Q-Networks (DQN) which enables the agent to take better decisions. The work utilizes

Stable Baseline3 library which is a powerful tool in developing and testing reinforcement learning algorithms to train and test the DQN model.
The Structure of the paper is as follows: Section II gives an extensive survey of existing scheduling algorithms and also highlights on their performance, Section III delves deeper into the presented intelligent scheduling algorithm. Further, Section IV does an extensive analysis of results and a comparison of performance of proposed and existing algorithms. Lastly, section V concludes the work and its future directions.

LITERATURE REVIEW

A huge amount of work has been carried out by researchers in recent years to considering crucial factors like energy consumption, delay, coverage, sensor deployment and accuracy [10]. A survey has been carried out addressing these challenges.

The Paper [11] highlights the importance of energy-efficient data collection techniques to extend the lifetime of WSN nodes, which are typically small and battery-powered. Different approaches and algorithms for energy-efficient data collection in IoT-WSN systems are, with a focus on preserving energy and aggregating data effectively. The authors also discuss the importance of IoT in WSN presenting major classifications and reviews of energy-saving technologies. The paper [2] proposes a Federated Deep Reinforcement Learning (FDRL) approach for adaptive and energy-efficient routing. FDRL facilitates decentralized decision-making and adaptive routing in dynamic networks by harnessing both local and global coordination. Simulations depict the effectiveness of the FDRL routing in terms of packet loss, latency, energy efficiency, and scalability. The proposed method works better than traditional protocols and other present routing plans. The CCM-RL protocol is compared with earlier protocols, demonstrating superior performance with respect to average number of active sensor nodes, coverage rate, and energy consumption. The algorithm in [12] leverages reinforcement learning, specifically Nash Q-Learning, to enable each sensor node to learn its optimal action to increase coverage and maintain connectivity. Every sensor node independently gets educated its optimal action to minimize the total number of activated nodes in each scheduling round while preserving coverage and connectivity.

The paper in [6] provides an open-source Python library that provides reliable and consistent implementations of deep reinforcement learning (RL) algorithms. SB3 includes implementations of seven commonly employed model-free deep RL algorithms. Overall, Stable-Baselines3 is designed to be a reliable, user-friendly, and well-documented library for deep reinforcement learning, making it easier for researchers and practitioners to develop and compare RL algorithms. The goal of the paper [13] is to schedule the sleep and wake-up times of sensor nodes to maximize their energy efficiency and extend the network's lifespan without compromising data packet transfer efficiency. The proposed method leverages reinforcement learning to allow each sensor node to independently decide its activity (transmission, tuning, or sleep) in a decentralized manner. The algorithm aims to minimize energy consumption by maintaining nodes in sleep mode as long as possible without lessening efficiency of data packet transfer. Each node independently decides its activity, which reduces the need for synchronization and communication overhead. The simulation results demonstrate the effectiveness of the proposed algorithm under various situations, showing improved performance in terms of energy savings and network lifespan. The presented paper in [9] uses Q-Learning, a reinforcement learning technique, to enable sensor nodes to autonomously decide their tasks (e.g., sleep, acquire, transmit, receive, forward, aggregate) based on environmental and application requirements. The algorithm minimizes energy consumption by allowing nodes to make autonomous decisions based on their present state and environment. Sensor nodes can adjust to changes in the environment and network conditions, ensuring efficient task scheduling. The simulation results show that the proposed algorithm majorly reduces energy consumption and maintains a high quality of service, with action failure rates below 10%.

The algorithm in [14] adapts to temperature variations, which significantly affect the performance of WSNs. Nodes exposed to higher temperatures consume more energy, and the algorithm helps mitigate this by adjusting the sleep schedules. The simulation output demonstrate the effectiveness of the proposed method under various conditions, showing improved performance with respect to energy savings and network lifespan. The proposed method shows qualified performance improvements in simulations, making it a promising approach for WSNs. In [15] the paper discusses the challenges and solutions for implementing time-sensitive networking (TSN) in wireless environments, specifically IEEE 802.11 networks and introduces traffic scheduler named WISE (Wireless Intelligent Scheduler). The method shows up to 99.9% of latency requirements under different wireless communication situations. The

processing delay is effectively kept within the specified time limits, ensuring thr scalability of TSN streams. WISE adjusts to variations in wireless channel conditions and satisfies the latency demands of time-sensitive streams. The study [16] introduces an innovative approach by integrating Deep Q-Network (DQN) with Graph Neural Network (GNN) to improve user base station association. The DQN-GNN method effectively captures intricate relationships within wireless networks, resulting in more precise and efficient user associations. It outperforms other techniques in terms of average rewards, returns and success rates.The authors in [10] have proposed a resource allocation method which carries out scheduling of node and multi target tracking accurately in mobile networks. An infinite horizon Markov decision process is applied to solve dynamic tracking problem by considering Energy consumption and delay as the major parameters. Model free reinforcement learning is employed to get accurate tracking actions which resulted in significance reduction in energy consumption and faster convergence compared to baseline approaches. Few more performance metrics like tracking accuracy, robustness and reliability could have been considered. Also, practical challenges of deployment are not discussed. The primary goal in [4], is to structure an energy-preserving Medium Access Control (MAC) protocol using Reinforcement Learning (RL) to increase the network lifetime. **QL-MAC Protocol**: The proposed QL-MAC protocol is gained from Q-learning, which iteratively adjusts the MAC parameters through a trial-and-error process to achieve a low energy state. QL-MAC adjusts the WSN node duty-cycle, reducing energy usage without negatively affecting other network parameters. Based on traffic predictions and transmission state of neighboring nodes the sleeping and active periods of radio channel are adjusted by the protocol. The protocol adjusts radio sleeping and active periods based on traffic predictions and the state of neighboring nodes. QL-MAC adapts to topological and external changes, providing a flexible and efficient solution for WSNs. Extensive experiments on off-the-shelf devices and large-scale simulations show that QL-MAC significantly reduces energy consumption while maintaining network performance. The protocol demonstrates higher performance in terms of Packet Delivery Ratio (PDR) and energy consumption in comparison to conventional MAC protocols.

The research work carried out in [17] explores the use of reinforcement learning to optimize the scheduling of wireless power transfer (WPT) and data transmission in Wireless Powered Sensor Networks (WPSNs). The scheduling problem is formulated as an MDP, considering sensor's energy consumption and data queue states. An optimal scheduling strategy with full-state information is also investigated to present the lower bound of data packet loss. The proposed reinforcement learning algorithm significantly reduces network packet loss by 60% and increases network output by 67% compared to existing non-MDP greedy approaches. The performance loss due to the lack of full-state information is less than 4.6%. The paper [18], highlights the use of deep reinforcement learning (DRL) to optimize task scheduling in edge computing environments and to maximize the long-term task satisfaction degree (LTSD) by efficiently scheduling tasks to virtual machines (VMs) at the edge server. The paper leverages a policy-based Reinforcement algorithm and a fully-connected neural network (FCN) to solve the task scheduling problem, addressing both time scheduling (task execution order) and resource allocation (VM assignment). Extensive simulations demonstrate that the proposed algorithm achieves better task satisfaction compared to baseline task scheduling algorithms. The paper in [19] explores the use of Support Vector Machine (SVM) models for packet scheduling in wireless communication networks. The performance of SVM-PFS is analyzed for various system and machine learning parameters, such as the average window, signal-to-interference ratio (SIR), and the degree of the polynomial. The polynomial SVM-PFS method provides better user selection accuracy and throughput fairness compared to the linear SVM-PFS method, the paper demonstrates that SVM models can effectively improve packet scheduling in wireless communication networks, with polynomial SVM-PFS showing superior performance.

After an extensive survey, it can be found that an intelligent algorithm can be the solution for scheduling problem in wireless sensor networks with dynamic conditions. The intelligent scheduling algorithm should have lesser time latency and alert accurate sensor node for a sensed action or event. The energy optimization can also be achieved by making the sensor nodes to be in the sleep mode until the agent interacts with the particular sensor node.

PROPOSED METHOD

Deep Q-Network (DQN) Model Architecture

The DQN model is critical for enabling the agent to learn and make decisions within the WSN environment, providing the foundation for reinforcement learning based scheduling strategies [20] . Figure 4 shows the process of DQN. The novelty of DQN scheduling in the proposed work is that the agent senses the actions and selects the appropriate sensor node to which the action is related. Also, the sensor node remains in sleep mode till it gets intimated by the agent regarding the action occurred. The intelligence of the Deep Q-Network lies in choosing the action with the highest reward based on epsilon greedy policy. Further, the loss computation is performed as feedback value to train the model in future episodes. Finally, the model is saved and loaded.

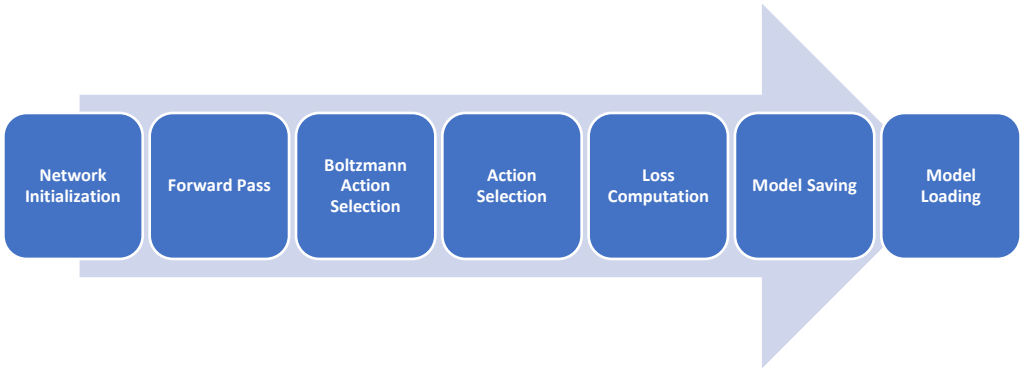

Figure 4: The DQN architecture and associated methods used in WSN environment.

Network Initialization: The neural network is initialized with an input layer that matches the observation space's shape, followed by two hidden layers with 64 neurons each and an output layer that corresponds to number of actions. The network uses the Tanh activation function. **Forward Pass:** The method flattens the input tensor and passes it through the network to compute the Q-values for each action. **Boltzmann Action Selection:** This method applies a softmax function to the Q-values, which converts them into probabilities. It then selects an action based on these probabilities. **Action Selection :** The method utilizes the Boltzman action selection method to choose an action given the current observation, applying a specified temperature for exploration. **Loss Computation:** This method calculates the loss between predicted Q-values and target Q-values using the smooth L1 loss function (Huber loss). The targets are computed based on the rewards and the maximum Q-value for the next state, adjusted by the discount factor. **Model Saving:** This method serializes the model's parameters and saves them to a specified file path using msgpack format. **Model Loading:** This method de-serializes and loads the model's parameters from a specified file path, restoring the network's state.

This work includes the implementation of several algorithms designed to optimize sensor energy consumption and improve state estimation within the WSN environment.
Round Robin Scheduler: A simple scheduler that systematically cycles through the available actions.
Random Scheduler: A baseline scheduler that selects actions randomly from the available action space.
Q-Learning Model: A traditional reinforcement learning approach that selects actions based on learned Q-values.
Deep Q-Network Model (DQN): A more advanced model that uses deep neural networks to make complex scheduling decisions. This model is the primary focus of the paper. [20]
Stable Baselines3 DQN Model: An implementation of the DQN model using the Stable Baselines3 library, providing a robust and scalable approach to training and evaluation.

Wireless Sensor Network Environment setup

A customized WSN environment is setup using Gym Library API standard which is a major toolkit in the field of Reinforcement Learning. This library provides a way to standardize environments for developing and evaluating algorithms. A class names WSNEnvironment inheriting gym.Env makes the network compatible with Gym library. The sensor positions are generated using Poisson distribution. The Algorithm is as follows.

Step 1	Define a class “WSNEnvironment” that inherits from gym.Env.
Step 2	Set up the environment with no. of sensors, sensor coverage, maximum steps and threshold probability. Call a method to generate sensor positions.
Step 3	Define a observation space as box with shape (self.num_points,3) representing rows and columns. Define the action space as number of points.
Step 4	Internal state variables like step count, information dictionary, event variable and generated events counter is initialized. Define a reset method to start the environment at the start of each episode.
Step 5	Initialize each sensor’s event buffer with zero value. Generate a random event based on a probability threshold. If the even occurs, increment the event counter and generate its location.
Step 6	Calculate the distance between sensors and the event location and update the event buffer for sensors within coverage. Consider the event buffer for selected sensor and update the observation space.
Step 7	Calculate the reward based on captured events and updates the event buffers.

The distribution of sensor nodes is shown in figure 5.

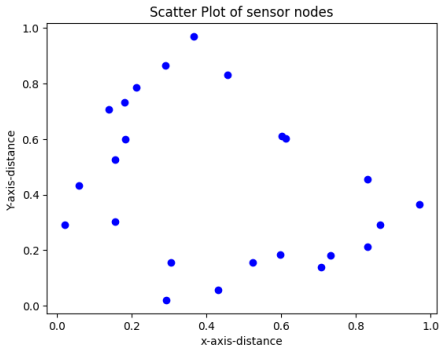

Figure 5: Sensor deployment Scatter plot

PERFORMANCE EVALUATION

The analysis offers insights into the relative performance of each scheduling strategy. The work has been carried out in Jupyter notebook. The work provides detailed analysis of the performance of each algorithm, including: Average Reward Comparisons: Evaluating the average rewards obtained by different schedulers over multiple episodes. Boxplot Analysis: Visualizing the distribution of rewards across different scheduling strategies. Latency Analysis: Assessing the time latency of captured events to evaluate the efficiency of each scheduler.

Random Scheduler Simulation

The Random scheduler is a non-deterministic method in which every action is selected randomly. The name Random scheduler is due to this random selection. Also it is simple to implement and requires no complex logic. Since all actions are equally likely to be selected it shows fairness in its execution. The Simulation of Basic Random scheduler for WSN environment comprises of these steps. The Simulation parameters are initialized with number of episodes equal to 30. An episode is a single run of environments from start to finish. An empty list to store scores of each episode is created. The score is calculated for each episode by resetting the environment at the beginning of each episode. Later, adds the reward obtained from the action to the score variable, accumulating the total score for the episode. Finally, calculates the Average Score of the total episodes.

Step 1	Number of Episodes: The Random scheduler algorithm is run for 30 number of episodes. At the starting point of each episode the environment is reset.
Step 2	The algorithm selects an action randomly from the considered action space.
Step 3	At every step, the selected random action is executed inside the environment and a reward is added to the episode’s score.
Step 4	Until the termination or truncation condition is met, the loop continues and its final score is recorded. The scores of all episodes and its average are stored and displayed.

The Limitations of Random scheduler are, it 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. A sample set of results obtained are shown below.

Episode:1 Score:294.47 {'captured': 703, 'non-captured': 6, 'sensor 0': set(), 'sensor 1': set(), 'sensor 2': set(), 'sensor 3': set(), 'sensor 4': set(), 'sensor 5': {(1, 999)}, 'sensor 6': set(), 'sensor 7': set(), 'sensor 8': set(), 'sensor 9': {(1, 999)}, 'sensor 10': set(), 'sensor 11': set()}
Episode:2 Score:280.14 {'captured': 671, 'non-captured': 5, 'sensor 0': set(), 'sensor 1': set(), 'sensor 2': set(), 'sensor 3': set(), 'sensor 4': set(), 'sensor 5': {(1, 988)}, 'sensor 6': set(), 'sensor 7': set(), 'sensor 8': set(), 'sensor 9': {(1, 988)}, 'sensor 10': set(), 'sensor 11': set()}

The above output depicts the detailed information of captured events, the non-captured events and the buffer status of each sensor. The number of non-captured events means, the events that were generated but are not captured by the sensors. The average Score is obtained as 290.4370595541624.

Calculation of Time Latency in Random Scheduler

Latency is defined as the difference between the current step count and the event occurrence time for each event captured by the selected sensor. The calculation and visualization of the time latency for events captured by the algorithm can be achieved by following these steps: Set up and reset the environment to prepare for the simulation. While running the Random Scheduler with all episodes, time latency data has to be collected. Further, Scheduler's performance is examined by calculating the average, minimum and maximum latency. Figure 6 shows the evolution of latency for captured events across the episode, with labeled axes and a legend. The average, maximum and minimum latencies for an episode are computed and analyzed showing the details of the performance of Random scheduler in terms of Time efficiency. Average Time Latency is 4.65 min: 1.00 max: 45.00 from the obtained Latency plot.

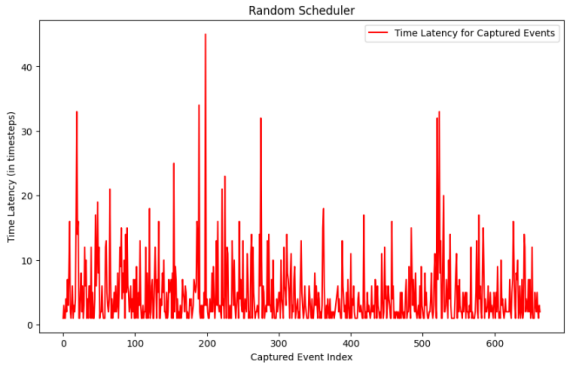

Figure 6:Time Latency of captured events in Random scheduler

Round Robin Scheduler

The steps involved in Round robin Scheduler algorithm are Initialization of number of episodes, sensors, and list to store scores. In this algorithm, each sensor is selected in a sequential order in a circular fashion. After each episode rewards are collected. The score of each episode is calculated and finally its average is computed. The average score is 307.91904931583684. Time Latency Computing for Round Robin Scheduler. Initially, environment is setup and reset. Every sensor node is selected sequentially to capture events. Time latency data is collected and average minimum and maximum latency are calculated. Finally, the performance of the algorithm is evaluated. Figure 7 shows the Time latency for Round Robin scheduler. Its Average Time Latency is 3.90 min: 1.00 max: 33.00.

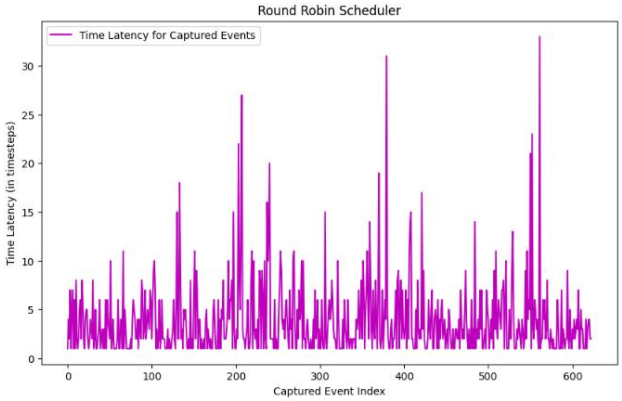

Figure 7: Time Latency of captured events in Round Robin scheduler

Q-Learning Agent Implementation for WSN Environment.

A Q-learning agent is defined for interacting with Wireless Sensor Network environment. Reinforcement learning is used to implement agent. The Process includes: Initializing the Q-learning agent with parameters like learning rate, initial epsilon, epsilon decay rate, final epsilon value and discount factor. The state-action values are also stored. The agent explores the environment with a defined probability epsilon and acquires the knowledge by selecting the action with the maximum Q-value. Based on the received rewards and the temporal difference between the predicted and actual values, Q-value is updated. Exploration over time is reduced by Epsilon decay mechanism, which encourages the agent to exploit learned strategies more as training progresses.

Hyper parameter Setup for Q-learning Agent:

To train the Q-learning agent, hyper parameters are set up in WSN. The important hyper parameters included are: Learning Rate which controls how much the agent adjusts its Q-values in response to new information. Number of episodes that defines how many episodes the agent gets trained for. Initial epsilon represents the initial probability of selecting a random action (exploration). Epsilon Decay is used for reducing the exploration over time to encourage more exploitation. Final epsilon depicting the minimum exploration rate. Using these hyper parameters as shown in Table 1, an instance of WSNEnvironment is created and the agent is prepared for the future training process.

Table 1: Hyperparameter setting values

Sl.No.	Hyper parameter	Set Value
1	Learning rate	0.01
2	No. of episodes	10000
3	Start of epsilon	1.0
4	Epsilon decay	0.0002
5	Final epsilon	0.1

Training the Q-learning Agent

The Q-learning agent is trained in WSN environment. The total number of episodes considered in 10,000. For every new episode the environment is reset. Within every episode, the agent selects actions based on the current observation, receives rewards and updates its Q-values accordingly. After the episode ends, agent's state is updated and epsilon decay takes place. The exploration rate is gradually reduced encouraging the agent to exploit learned behaviors more over time.

Q-Learning Scheduler Evaluation

The performance of the Q-learning Scheduler in WSNEnvironment is evaluated by applying the following process. The evaluation parameters are initialized. After every episode, the environment is set up and agent selects the actions based on the learned policy. The rewards are collected until the episode ends. The Score is recorded along with captured and non captured events. The average score is calculated show casing the overall measure of the Q-learner's scheduler. The average score is found to be 210.4030243543611.

Time Latency Calculation for Q-Learning Scheduler

The Time Latency for events captured by the Q-Learning scheduler follow the process: Environment Set up, Execution of the Q-Learning scheduler, Collection of time latency data by finding the difference between the current step count and the event occurrence time for each event captured by the selected sensor. Later, the average, minimum and maximum latency is computed to evaluate the performance of Q-scheduler. Further, visualization of time latency for captured events is performed. Average Time Latency is found to be 4.95 min: 1.00 max: 36.00 and is shown in Figure 8.

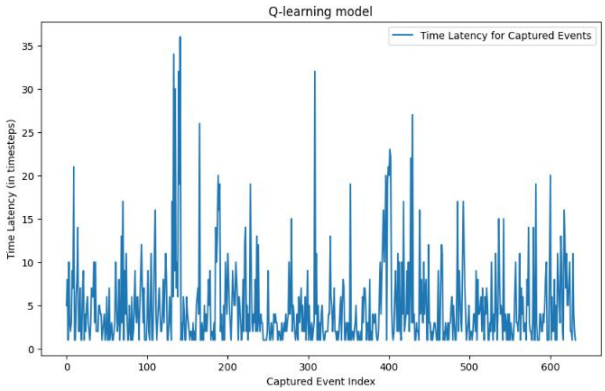

Figure 8: Time Latency of captured events in Q-Learning model

Hyperparameter setup for Deep Q-Network (DQN)

Hyperparameters are user defined parameters that are not learned by the neural network during training. These are set by the user prior to training and manipulate the training of an artificial neural network. High quality results can be obtained by tuning the hyperparameters as depicted in Table 2.

Table 2: Hyperparameter values settings

Hyperparameter	**Vlaue**
Discount Rate √	0.99
Batch size	32
Buffer size	1x 10^6
Minimum Replay size	1000
Epsilon start	1.0
Epsilon End	0.1
Epsilon Decay	3x10^6
Target Update Frequency	1000
Learning Rate	5x10^{-5}

Deep Q-Network (DQN) Training Process

The important steps involved are:

Replay and Reward Buffers Initialization: The buffer stores transitions (state, action, reward, next state) with a maximum length. Network Initialization: The DQN network is initialized and cloned with the target network to compute stable targets during training. Both the networks are synchronized. Optimizer setup: An Adam optimizer is initialized to update the network weights based on the computed loss [20]. Replay Buffer population: The replay buffer is filled with random actions until it reaches the minimum required size. Main Training loop: The agent interacts with the environment, selects actions and stores transitions in the Replay buffer. After each step, the agent accumulates rewards, and the training loop updates the Q-network by sampling a batch of transitions from the replay buffer. The loss is computed and back propagation is performed to update the network weights. The target network is updated periodically to match the online network. Training progress is logged, and the model is saved at specified intervals. The training process enables the DQN agent to learn and improve its policy over time, optimizing sensor scheduling in the WSN environment. Average reward got is 288.7955347162685 as shown in Figure 9. A graph plotted between number of episodes and number of steps is depicted in figure 10.

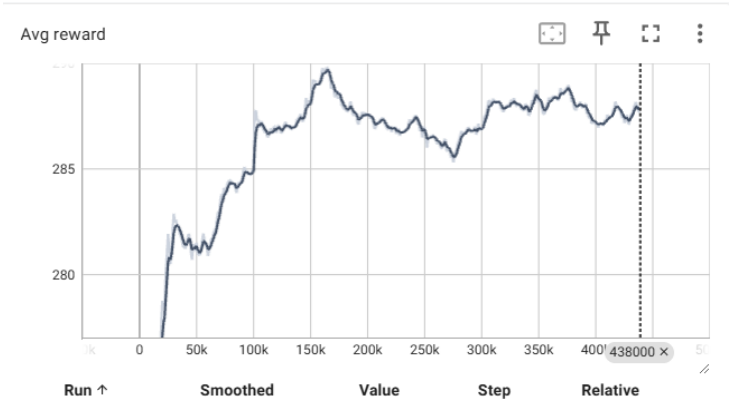

Figure 9: Average Reward obtained

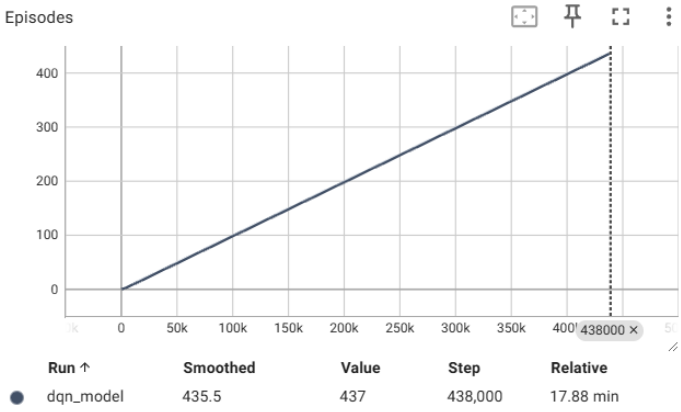

Figure 10: Episodes versus number of steps of DQN scheduler

Evaluation of the Deep Q-Network (DQN) Scheduler

The performance of DQN scheduler [21] is evaluated using these processing steps: Initialization of evaluation parameters, execution of DQN scheduler, recording the scores and average score calculation. Average score is found to be 295.6934780476084.

Time Latency computing for DQN Scheduler

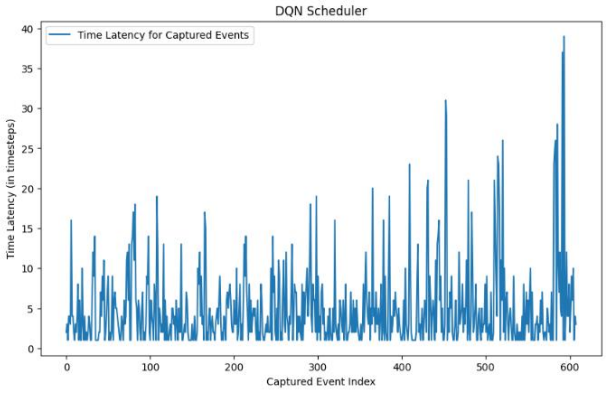

Figure 11: Time Latency for captured events of DQN Scheduler

The average Time Latency is calculated as 4.97 min: 1.00 max: 51.00 as depicted in Figure 11.

Comparison of Scheduler Performance

A comparison and visualization of different scheduling strategies in WSN environment by plotting the average rewards obtained per episode is shown in fig 12 between Random scheduler, Round robin scheduler and DQN scheduler. The x-axis represents sequence of episodes and average rewards per episode on the y-axis. The plot depicts a clear visual comparison of how each scheduling strategy performs over time, allowing for an easy assessment of which approach yields better results in WSN environment. 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 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 in terms of Latency.

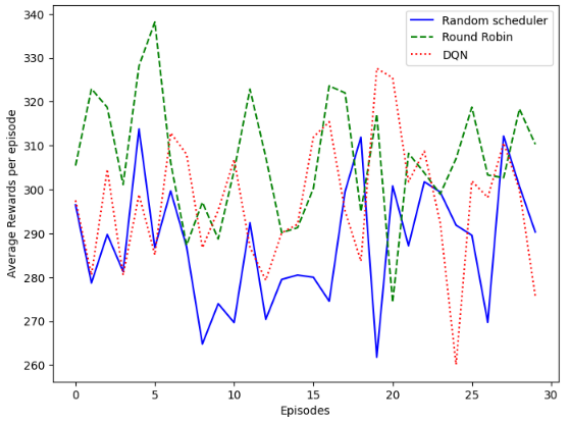

Figure 12: Comparison of Scheduler performance between different schedulers

Boxplot Comparison of Scheduling Algorithms

The fig 13 shows a boxplot to compare the distribution of rewards obtained by different scheduling algorithms in WSN environment. The Process includes:

Data preparation: A 2D array is created where each row corresponds to the rewards per episode for a specific scheduling algorithm like Random scheduler, Round robin scheduler and DQN scheduler. A boxplot is generated to visualize the distribution of rewards for each scheduling algorithm, providing insights into their performance visibility. The appearance of the boxplot is enhanced by customizing the colors and line widths of the boxes, whiskers, caps, medians and filters. The x-axis labels each scheduler, and the y-axis represents the average rewards per episode. A title is added to describe the comparison. The boxplot provides a clear visualization of the performance distributions for the different scheduling strategies, highlighting which algorithm tends to perform better in terms of consistency and average rewards.

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

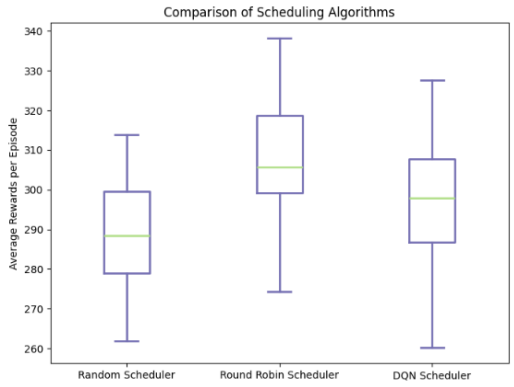

Figure 13: Box plot representation for performance of scheduling algorithms

Model Testing and Evaluation

The performance of saved Deep Q-Network (DQN) model in the WSN environment including the processing steps: Test function definition is the number of episodes for which model was run are 20. The saved DQN model is loaded from a specified path and instantiated on the CPU. The test model function is called to evaluate the DQN model's performance in the WSN environment and its rewards are listed in Table 3. The average reward over 20 episodes is 298.065879830779.

Table 3: Obtained Rewards for 20 Episodes

Episode	Reward
1	29306572058796535
2	307.1150884094625
3	288.1313834183529
4	312.08260533638474
5	295.1990196301991
6	305.1336945794142
7	296.43867058896
8	301.4231326957663
9	311.0925900587663
10	281.2853079559847
11	280.8378406590823
12	285.0297242206506
13	310.73913535838363
14	280.3127584529237
15	300.63909054247836
16	317.94242097771496
17	292.2499893654911
18	322.80223206434965
19	298.15714514738846
20	298.15714514738846

Latency Comparison of different Schedulers

The fig 14 shows the comparison between latency of different scheduling algorithms in the WSN environment by plotting 100 latency values for each scheduler. The process includes Data selection where it captures only 100 elements from the latency lists for each scheduler. A line plot is created to visualize the latency for each scheduler. The x-axis represents time steps and y-axis represents latency. The plot in fig 14 shows a visual comparison of the latency performance for each scheduling strategy highlighting which algorithm minimizes latency more effectively in the WSN environment. Table 4 highlights the score values of considered algorithms.

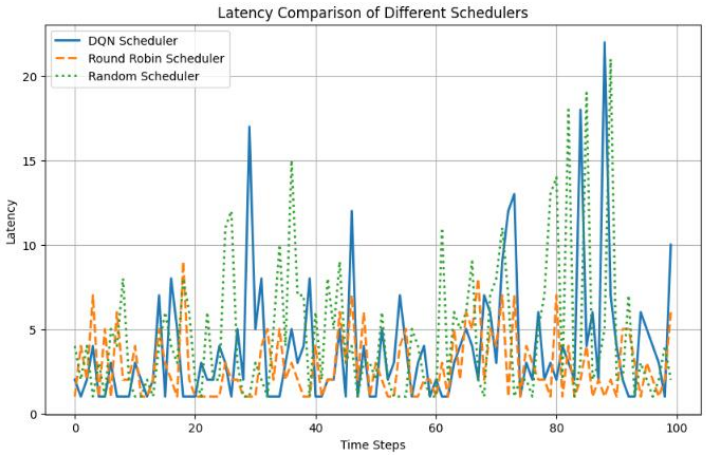

Figure 14:Latency comparison of Different Schedulers

Table 4: A comparison of Scores between considered algorithms

Episode	Score (Random)	Round	Q-learning	DQN	Baseline Random scheduler
1	294.7	307.55	195.20	317.39	275.74
2	280.14	316.21	221.41	318.97	301.44
3	297.97	332.17	220.41	280.31	297.23
4	317.22	338.10	219.18	306.78	303.62
5	295.62	303.77	223.29	285.92	292.09
6	294.87	317.65	227.47	290.34	301.74
7	309.23	305.08	228.49	292.45	308.89
8	308.17	310.78	220.06	290.94	270.44
9	297.88	290.98	182.57	298.34	309.49
10	274.62	324.74	206.45	291.20	272.44
11	259.37	292.46	179.74	309.01	301.67
12	282.32	299.81	214.12	291.49	299.89
13	271.43	308.06	230.74	285.59	300.89
14	320.18	304.81	190.84	303.72	274.73
15	290.77	307.84	219.59	299.44	291.23
16	273.54	320.88	223.69	292.20	281.70
17	266.48	300.80	207.02	301.31	295.45
18	284.21	311.15	186.43	315.13	306.05
19	285.44	306.31	208.75	314.48	308.89
20	308.89	328.80	239.73	313.26	300.79
21	311.81	304.72	194.97	298.04	303.45
22	273.34	289.71	217.09	298.28	306.60
23	273.80	301.71	178.68	282.60	258.54
24	314.93	326.38	220.29	284.51	277.85
25	282.42	282.51	251.21	293.67	271.86
26	284.50	311.15	180.51	271.38	299.72
27	271.92	295.88	208.73	307.50	301.49
28	297.95	310.12	204.80	263.70	293.83
29	306.94	311.67	210.36	295.19	289.00
30	282.68	275.77	234.21	277.70	283.27

Stable Baselines3 scheduler

The Stable- baselines3 is a package installed to provide the DQN algorithm and other tools for reinforcement learning [6]. Gymnasium toolkit is used for developing and comparing reinforcement learning algorithms. Poisson distribution is employed to generate sensor positions. A WSN environment is initialized and setup. The custom environment becomes a foundation for testing and developing intelligent scheduling strategies focusing on reinforcement learning applications. The step function defines the environment's response to an agent's action, including event generation, sensor selection, and reward calculation and termination condition checks. Placeholder methods are used to render and close the environment. The average score is 292.66682659870133.

Table 5: A Comparison of Average reward and Latencies

Algorithm	Avg Reward	Avg Time Latency	Min Latency	Max Latency
Random Scheduler	290.4370595541624	4.65	1.00	45.00
Round Robin Scheduler	307.91904931583684	3.90	1.00	33.00.
Q-Learning	210.4030243543611	4.95	1.00	36.00
DQN	295.6934780476084	4.97	1.00	51.0
SB3	281.2568456255548	3.24	1.00	31.00.

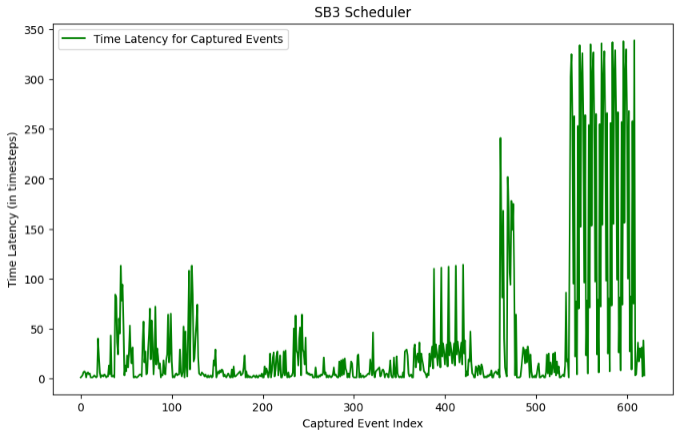

Figure 15:Time Latency for captured events of SB3 Scheduler

Time Latency of SB3 scheduler is shown in Figure 15. The average latency is 3.24 mi: 1.00 max:31.00. Overall, the Round Robin scheduler appears to be the most promising with the highest propensity for higher rewards, despite its greater variability. The Table 5 shows the comparison between Average rewards and latencies between the Proposed DQN scheduling and existing schedulers. It clearly shows that 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 results depict that the DQN with SB3 scheduler shows lesser latency in comparison to earlier scheduling methods.

CONCLUSION

In this article, a reinforcement learning based Deep Q Network scheduling is proposed for sensor node scheduling in wireless sensor networks. Initially, a customized simulation environment tailored to our specific needs using the Gym API, a well-established for reinforcement learning environments was constructed. Further, Random scheduler, Round robin scheduler are implemented for the same WSN environments. Later, DQN scheduler in implemented by integrating Baseline 3 for scheduling operation. The simulation experiments validate the effectiveness of the proposed algorithm. The algorithm's validation process involved several key steps, implemented to meet high standards of rigor and reproducibility. 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. 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 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.

REFERENCES:

[1] X. Wang, H. Chen, and S. Li, "A reinforcement learning-based sleep scheduling algorithm for compressive data gathering in wireless sensor networks," *J Wireless Com Network*, vol. 2023, no. 1, p. 28, Mar. 2023, doi: 10.1186/s13638-023-02237-4.

[2] S. S. Suresh, V. Prabhu, V. Parthasarathy, G. Senthilkumar, and V. Gundu, "Intelligent data routing strategy based on federated deep reinforcement learning for IOT-enabled wireless sensor networks," *Measurement: Sensors*, vol. 31, p. 101012, Feb. 2024, doi: 10.1016/j.measen.2023.101012.

[3] S. Redhu, R. Mahavar, and R. M. Hegde, "Energy-efficient wake-up radio protocol using optimal sensor-selection for IoT," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2018, pp. 1–6. doi: 10.1109/WCNC.2018.8377328.

[4] C. Savaglio, P. Pace, G. Aloi, A. Liotta, and G. Fortino, "Lightweight Reinforcement Learning for Energy Efficient Communications in Wireless Sensor Networks," *IEEE Access*, vol. 7, pp. 29355–29364, 2019, doi: 10.1109/ACCESS.2019.2902371.

[5] A. A. Deshpande, F. Chiariotti, and A. Zanella, "Energy-Efficient Internet of Things Monitoring with Content-Based Wake-Up Radio," Dec. 07, 2023, *arXiv*: arXiv:2312.04294. doi: 10.48550/arXiv.2312.04294.

[6] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations".

[7] S.-Y. Wang and C.-D. Lin, "Using deep reinforcement learning to train and periodically re-train a data-collecting drone based on real-life measurements," *Journal of Network and Computer Applications*, vol. 221, p. 103789, Jan. 2024, doi: 10.1016/j.jnca.2023.103789.

[8] Z. Zhang, L. Shu, C. Zhu, and M. Mukherjee, "A Short Review on Sleep Scheduling Mechanism in Wireless Sensor Networks," in *Quality, Reliability, Security and Robustness in Heterogeneous Systems*, vol. 234, L. Wang, T. Qiu, and W. Zhao, Eds., in Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 234. , Cham: Springer International Publishing, 2018, pp. 66–70. doi: 10.1007/978-3-319-78078-8_7.

[9] C. Cirstea, R. Davidescu, and A. Gontean, "A reinforcement learning strategy for task scheduling of WSNs with mobile nodes," in *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, Rome, Italy: IEEE, Jul. 2013, pp. 348–353. doi: 10.1109/TSP.2013.6613950.

[10] X. Zou, L. Li, H. Du, and L. Zhou, "Intelligent Sensing and Computing in Wireless Sensor Networks for Multiple Target Tracking," *Journal of Sensors*, vol. 2022, pp. 1–11, Apr. 2022, doi: 10.1155/2022/2870314.

[11] Kamal gulati, Raja Sarath Kumar, " A review paper on wireless sensor network techniques in Internet of Things (IoT)", https://www.researchgate.net/publication/351725599, DOI:10.1016/j.matpr.2021.05.067 .

[12] A. Sharma and S. Chauhan, "A distributed reinforcement learning based sensor node scheduling algorithm for coverage and connectivity maintenance in wireless sensor network," *Wireless Netw*, vol. 26, no. 6, pp. 4411–4429, Aug. 2020, doi: 10.1007/s11276-020-02350-y.

[13] P. Verma, A. Dumka, D. Vyas, and A. Bhardwaj, "Reinforcement Learning based Node Sleep or Wake-up Time Scheduling Algorithm for Wireless Sensor Network," *Int J Math, Eng, Manag Sci*, vol. 5, no. 4, pp. 707–731, Aug. 2020, doi: 10.33889/IJMEMS.2020.5.4.057.

[14] P. S. Banerjee, S. N. Mandal, D. De, and B. Maiti, "RL-Sleep: Temperature Adaptive Sleep Scheduling using Reinforcement Learning for Sustainable Connectivity in Wireless Sensor Networks," *Sustainable Computing: Informatics and Systems*, vol. 26, p. 100380, Jun. 2020, doi: 10.1016/j.suscom.2020.100380.

[15] H. Kim, Y.-J. Kim, and W.-T. Kim, "Deep Reinforcement Learning-Based Adaptive Scheduling for Wireless Time-Sensitive Networking," *Sensors*, vol. 24, no. 16, p. 5281, Aug. 2024, doi: 10.3390/s24165281.

[16] I. Alablani and M. J. F. Alenazi, "DQN-GNN-Based User Association Approach for Wireless Networks," *Mathematics*, vol. 11, no. 20, p. 4286, Oct. 2023, doi: 10.3390/math11204286.

[17] K. Li, W. Ni, M. Abolhasan, and E. Tovar, "Reinforcement Learning for Scheduling Wireless Powered Sensor Communications," *IEEE Trans. on Green Commun. Netw.*, vol. 3, no. 2, pp. 264–274, Jun. 2019, doi: 10.1109/TGCN.2018.2879023.

[18] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep Reinforcement Learning-Based Task Scheduling in IoT Edge Computing," *Sensors*, vol. 21, no. 5, p. 1666, Feb. 2021, doi: 10.3390/s21051666.

[19] S. Bhandari, H. P. Zhao, H. Kim, P. Khan, and S. Ullah, "Packet Scheduling Using SVM Models in Wireless Communication Networks".

[20] X. Fu and J. G. Kim, "Deep-Q-Network-Based Packet Scheduling in an IoT Environment," *Sensors*, vol. 23, no. 3, p. 1339, Jan. 2023, doi: 10.3390/s23031339.

[21] M. Sewak, "Deep Q Network (DQN), Double DQN, and Dueling DQN: A Step Towards General Artificial Intelligence," 2019, pp. 95–108. doi: 10.1007/978-981-13-8285-7_8.