

# Deep Q-network Reinforcement Learning-based Routing Adjustment Mechanism for Carbon Footprint Reduction

Tin-Yu Wu,<sup>1\*</sup> Chu-Fu Wang,<sup>2</sup> Wei-Che Chien,<sup>3</sup> Yi-Kai Chen,<sup>4</sup> and Deng-Ying Liu<sup>5</sup>

<sup>1</sup>Department of Computer Science and Information Engineering,  
National Yunlin University of Science and Technology, Taiwan, R.O.C.

<sup>2</sup>Department of Computer Science and Artificial Intelligence, National Pingtung University, Taiwan, R.O.C.

<sup>3</sup>Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan, R.O.C.

<sup>4</sup>Department of Management Information Systems, National Pingtung University of Science and Technology,  
Taiwan, R.O.C.

<sup>5</sup>Department of Engineering Science, National Cheng Kung University, Taiwan, R.O.C.

(Received April 2, 2026; accepted August 31, 2026)

**Keywords:** routing protocol, deep Q network, quality of service, network energy saving and carbon reduction, deep learning

The flourishing development of network applications and humans' increasing needs on the internet have contributed to a significant increase in not only global data traffic volume <sup>(1)</sup> but also carbon emissions generated from network services year after year. Therefore, with the public awareness of environmental sustainability, issues related to network energy saving and carbon reduction are receiving considerable attention. At present, the purpose of most studies related to carbon reduction is to reduce energy consumption to reduce carbon emissions. However, reducing carbon emissions is not the only way to save energy. For carbon reduction, the carbon intensity of data transmission across networks can be measured for data transmission strategy adjustment. However, a minimum level of the quality of service (QoS) should be ensured while choosing data transmission paths with low carbon emissions, in case an overemphasis on reducing carbon emissions compromises the QoS. In this study, we organize the connections between QoS and user experience and identify several significant metrics that most affect user experience and their smallest acceptable values. To achieve a routing adjustment mechanism for guaranteeing QoS and carbon reduction, the median life cycle carbon emissions provided by the Intergovernmental Panel on Climate Change (IPCC) is used as the carbon intensity indicator (CII) to find the route with the lowest carbon intensity within the median range.

## 1. Introduction

With the rising awareness of sustainable development and environmental issues, eco-friendly products or solutions have been presented in a wide variety of fields in recent years. Instead of striving for maximum performance, these products focus on energy saving and carbon reduction.

---

\*Corresponding author: e-mail: [tyw429@gmail.com](mailto:tyw429@gmail.com)  
<https://doi.org/10.18494/SAM6365>

Energy saving means reducing energy consumption in order to reduce carbon emissions without compromising the performance of products or solutions. Carbon reduction means lowering the amount of carbon emissions while maintaining a product's performance. Both energy saving and carbon reduction aim to decrease carbon emissions and create a sustainable environment. In the field of networks, the increasing data traffic as well as multiple and abundant network services have continuously increased the energy consumption of network devices.<sup>(1)</sup> Thus, issues about network energy saving and carbon reduction are drawing much more attention from the public. At present, either for commercial products or research fields, many more discussions are being made for energy saving rather than for carbon reduction. Therefore, this study will be concentrated on carbon reduction, that is, how to reduce carbon emissions without prioritizing energy-saving techniques.

In studies up to now, carbon-aware routing and network management have been investigated from different perspectives. However, the approaches reviewed in this study do not explicitly integrate carbon intensity with multiple QoS indicators, including latency, jitter, and packet loss rate, within a unified deep Q-network (DQN)-based routing framework. To address this gap, in this study, we develop a DQN-based routing adjustment mechanism that jointly incorporates carbon intensity and QoS indicators into routing decisions. The main contribution of this study is the design of a unified DQN-based routing framework in which carbon intensity, latency, jitter, and packet loss rate are jointly represented in the routing state and reward evaluation. This framework enables the routing policy to reduce carbon intensity while maintaining predefined QoS requirements under dynamically changing network conditions.

## **2. Background and Related Work**

In this section, we present the background and related work underlying the proposed routing adjustment mechanism. In Sect. 2.1, the carbon intensity of different electricity generation methods is introduced. Section 2.2 contains a review of conventional path-planning algorithms and DQN-based routing approaches, while in Sect. 2.3, the fundamental concepts of reinforcement learning and DQNs are described. Related studies on carbon-aware networking are discussed in Sect. 2.4, and the relationship between QoS indicators and user experience is summarized in Sect. 2.5. Finally, in Sect. 2.6, the limitations of existing approaches are identified and the differences between them and the proposed method are clarified.

### **2.1 Carbon intensity of electricity generation**

Networks require electricity to function; various fuels produce different amounts of carbon emissions during the generation of electricity. Network devices consume energy and contribute to significant carbon emissions,<sup>(2–4)</sup> among which information and communication technology (ICT) is responsible for approximately 2.7% and network services account for 13%.<sup>(1)</sup> Therefore, how to reduce carbon emissions while simultaneously maintaining the network performance is an important issue. To cope with climate change and to raise environmental awareness, low-emission sources of electricity, including wind energy, solar power, and nuclear energy, i.e.,

clean energy, have been developed in recent years. Electricity generation from clean energy reduces not only carbon emissions but also the impact of power generation on Earth. The Intergovernmental Panel on Climate Change (IPCC) was set up for assessing the science related to climate change for climate policies and legislation, and the IPCC publishes comprehensive scientific assessment reports on climate change periodically. As listed in Table 1, the IPCC's Fifth Assessment Report (AR5) released in 2014 gave the estimated median life cycle emissions for electricity production; this will be used as the carbon intensity indicator (CII) in this study for the routing protocol to calculate the weights related to carbon intensity.

## 2.2 Path planning and algorithm

For different network scales and scenarios, various algorithms and routing protocols are proposed. Common algorithms include the distance-vector (DV) algorithm and Dijkstra's shortest path first (SPF) algorithm.<sup>(5)</sup> However, with the increasing number of nodes and the development of AI, traditional algorithms lack flexibility and require corresponding adjustments. Therefore, smart AI-based routing protocols are becoming widely accepted. Zheng *et al.* proposed a routing optimization strategy that uses a DQN learning mechanism to allocate transmission paths for different priority levels of service requirements. The experiment results prove that their proposed mechanism, compared to OSPF and other machine learning strategies, obviously outperforms in terms of average delay and packet loss rate.<sup>(6)</sup> Ye *et al.* proposed a DQN-based intelligent QoS-optimized efficient routing algorithm for vehicular networks, showing its processing capability to intelligently select the optimal data transmission path by dynamically learning and adapting to the changes in the vehicular network.<sup>(7)</sup> In previous studies, the applicability of DQN and deep reinforcement learning (RL) to adaptive routing and multi-objective network decision-making problems was demonstrated.<sup>(6–14)</sup> For example, deep Q-learning has been applied to SDN-based data center routing to optimize delay, packet loss rate, and throughput.<sup>(12)</sup> DRL-OR addresses online routing for traffic flows with different service requirements,<sup>(13)</sup> while DeepQoS applies deep RL to QoS-aware routing in software-defined data center networks.<sup>(14)</sup> Those studies were primarily focused on service prioritization or QoS optimization. In contrast, the proposed method additionally incorporates carbon intensity together with latency, jitter, and packet loss rate into a unified DQN-based routing framework.

Table 1  
Carbon emissions of mainstream electricity generation methods, IPCC AR5.<sup>(2)</sup>

| Methods of generating electricity | Median life cycle emissions (unit: gCO <sub>2</sub> eq/kWh) |
|-----------------------------------|-------------------------------------------------------------|
| Coal combustion                   | 820                                                         |
| Natural gas combustion            | 490                                                         |
| Hydroelectricity                  | 24                                                          |
| Nuclear energy                    | 12                                                          |
| Wind energy <sup>1</sup>          | 11                                                          |
| Solar power <sup>1</sup>          | 41                                                          |
| Hydrogen energy                   | 24                                                          |
| Geothermal power                  | 38                                                          |

<sup>1</sup>Note: Wind power refers to on-shore wind power and solar energy refers to rooftop solar energy.

### 2.3 RL and DQN

RL, a type of machine learning, is different from supervised and unsupervised learning, since RL focuses more on decision-making enhancement and long-term outcomes. Through the agent interacting with an environment, RL learns to optimize strategies in order to maximize the cumulative reward.<sup>(15)</sup> RL consists of several key components.

- Agent: the decision-maker
- Environment: the external system the agent interacts with
- State: the current situation of the environment at a given time
- Action: the decisions the agent makes in the current state
- Reward: the numerical values that the agent receives after taking an action in a given state

For policy evaluation, dynamic programming (DP) is used in RL to break complex problems into smaller ones, solve the subproblems, and combine them to find the optimal strategies.

Q-learning, an RL algorithm based on function approximation, helps the agent learn the best actions within unknown environments and maximize the cumulative reward.<sup>(15)</sup> Q-learning includes the key components indicated as follows.

- State (S): the agent's current situation in the environment
- Action (A): the operations the agent chooses to enact in the environment
- Reward (R): the feedback received on the basis of the agent's action
- Q-value: the expected cumulative reward of taking a particular action in a specific state

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_{\alpha'} Q(S', \alpha') - Q(S, A)] \quad (1)$$

In Eq. (1),  $\alpha$  represents the learning rate,  $\gamma$  means the discount factor for future rewards,  $S'$  means the new state after taking an action, and  $\max Q(S', \alpha')$  means the maximum expected future reward for the next state.

Nevertheless, while dealing with large-scale or continuous state spaces, Q-learning has obvious limitations. The state space must be stored in the Q-table. As the state and action spaces become larger, the dimensions of the Q-table increase also, making it difficult to handle continuous state spaces and leading to a generalization ability to adapt quickly to a new environment. To solve the limitations mentioned above, DQN combines deep neural networks (DNNs) with Q-learning. The implementation of DQN requires three crucial mechanisms.

First, DQN uses the DNN as the function approximator to project high-dimensional input onto the action-value function. The DQN architecture comprises the input layer that receives the current state of the environment, the hidden layers that capture high-level features, and the output layer that provides the Q-values for all possible actions in the current state. This mechanism efficiently solves the curse of dimensionality to parameterize the Q-function.

Next, by introducing the experience replay mechanism, DQN stores experiences gathered from interactions of the agent with the environment ( $s, a, r, s'$ ) in the replay buffer and randomly samples the past experiences for training. Experience replay not only increases sample utilization, but also breaks correlations between sequential experiences, reducing the data deviation as well as enhancing the system stability.

Third, by adopting a target network topology, DQN uses a separate target network with the same architecture as the main network. The main network is used to choose the action and calculate the current Q-value. The target network is used to calculate the target Q-value, and the parameter  $\theta$  is periodically updated to match the weights of the main network. Such a design can reduce the target volatility and increase the training stability. Figure 1 shows the DQN architecture and how it works.

The loss function of DQN is defined as the square of the temporal difference error.

$$L(\theta) = E_{(s,a,r,s') \sim D} \left[ \left( y - Q(s,a;\theta) \right)^2 \right] \quad (2)$$

The target value  $y$  is computed as

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-), \quad (3)$$

where  $r$  means the immediate reward,  $\gamma$  means the discount factor ( $0 \leq \gamma \leq 1$ ),  $\theta$  means the main network's parameter,  $\theta^-$  means the target network's parameter, and  $D$  means the replay buffer. The training process of DQN begins by initializing the network parameters. The parameter  $\theta$  in the main network is initialized with random weights, and the parameter  $\theta$  in the target network is initialized with the same weights. An empty replay buffer  $D$  is created to store training data.

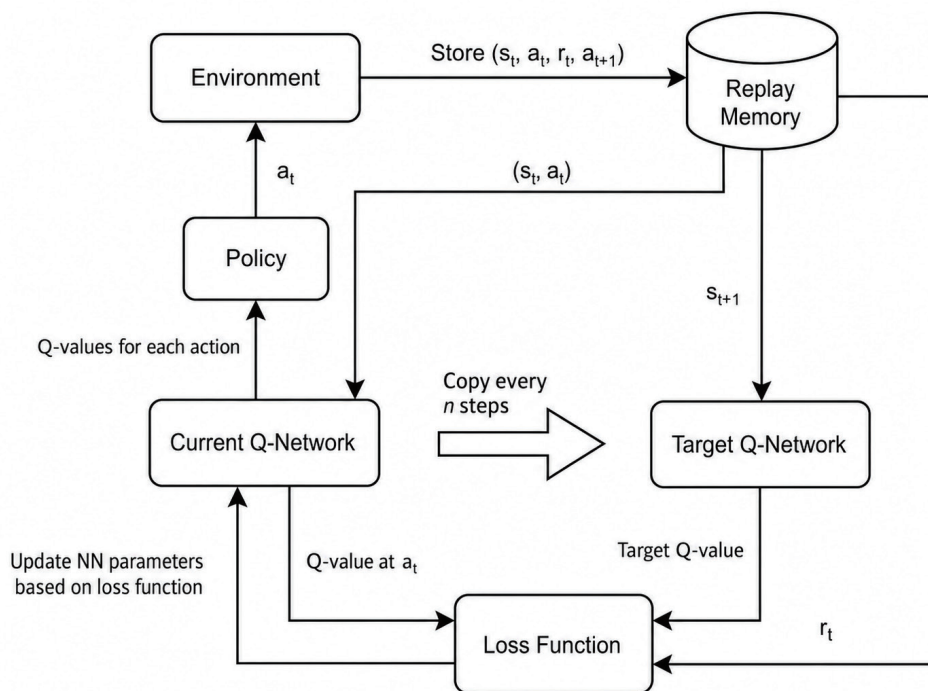


Fig. 1. DQN architecture.

In each round of training, on the basis of the current situation  $s$ , the agent selects an action in accordance with an  $\varepsilon$ -greedy strategy. After the action is executed, the agent receives an immediate reward, denoted as  $r$ . Then, the updated state, denoted as  $s'$ , and the transitions  $(s, a, r, s')$  are stored in the replay buffer. Next, the agent randomly samples a mini-batch of experiences from the buffer for training and computes the target value  $y$  and the loss function  $L(\theta)$ . The agent then performs a gradient descent step to update the parameters  $\theta$  in the main network. For the stability of the training, the parameters of the main network are updated periodically to the target network every  $C$  steps, so that  $\theta^- = \theta$ . This process is repeated until the agent is capable of choosing the right action in a given state.

Compared with traditional Q-learning methods, DQN has several significant advantages. DQN can handle high-dimensional raw inputs and has excellent feature extraction and representation learning capabilities. Also, DQN achieves good generalization performance, making it better generalized to unseen states. By employing innovative mechanisms such as experience replay and the use of a separate target network, DQN enhances the learning stability. Because of these advantages, DQN has good scalability and is especially suitable for solving large-scale decision-making problems with a large discrete action space, such as the path planning task in this study. Through reasonable network design and parameter adjustment, DQN can efficiently learn to optimize path planning based on the current state of the environment.

Recent studies have further improved deep reinforcement learning through enhanced experience replay and exploration strategies. MALight employs pressure information and attentive experience replay for traffic-signal control,<sup>(16)</sup> while DERLight adopts dual experience replay to improve learning efficiency.<sup>(17)</sup> In addition, a self-balancing exploration and exploitation strategy has been proposed to improve learning performance in sparse-reward tasks.<sup>(18)</sup> These studies demonstrate the importance of experience replay and exploration strategies for improving the adaptability of reinforcement learning in dynamic decision-making environments.

## 2.4 Related work in carbon-aware networking

With the increasing global emphasis on environmental sustainability, numerous studies have been dedicated to carbon-aware networking to investigate carbon emissions of data transmission on a network, to assess the effects of carbon emissions, and to reduce the carbon footprint. However, an issue worth discussing is that “saving energy” is a crucial aspect of reducing carbon emissions, but “using clean sources of energy” is also a way to achieve carbon reduction.<sup>(19,20)</sup> To achieve a green network, Hossain *et al.* proposed two metrics, the carbon emission factor (CEF) and nonrenewable energy usage percentage (NRE), to turn off unused nodes and dynamically adjust link rates using adaptive link rate (ALR) for energy-aware routing. They obtained carbon intensity data for different countries from previous studies and conducted all experiments on the Gigabit European Academic Network (GÉANT) topology with a realistic demand size. Together with the amount of traffic load for evaluation, a software-defined network approach that can be adjusted in accordance with CEF and NRE is established.<sup>(21)</sup>

In addition to the above-mentioned centralized routing decision-making mechanism in the SDN framework, some researchers focus on the connectivity needs and coordination between cross-network enterprises or network service providers. Every network-based enterprise builds or adopts its own enterprise intranet system, i.e., an autonomous system (AS), in accordance with their specific needs. Because of the concerns for information security and trade secrets, an enterprise usually does not release information about its own AS or routing method, or actively adapt itself to become compatible with other companies. As a result, the carbon intensity of data transmission lacks standardization and is often opaque among different ASs, making it difficult to regulate traffic flow on the basis of the carbon intensity. For this reason, Tabaeiaghdaei *et al.* proposed to exchange only the carbon footprint among ASs, without leaking the architecture of the AS, so as to regulate data transmission. In addition to guaranteeing the privacy of each AS, their method enables the AS to exchange the carbon intensity and to achieve carbon-aware global routing.<sup>(22)</sup> Tabaeiaghdaei *et al.* used tools such as electricity maps to provide geographic information and carbon intensity forecasts, and regulated data in accordance with the locations of access points and energy consumption parameters provided by the AS. Since the cross-domain network is decentralized and often uses protocols such as RIP, frequent topology changes or path reselection may prevent the network from converging to stability. To ensure network stability, the nodes adopt the carbon intensity forecasts and adjust the path hourly. Finally, each autonomous system makes the decision about which path to take in accordance with the carbon data exchange.

2.5 QoS and user experience

With the rapid advancement of Internet technology and applications, Internet access has become a necessity in every aspect of life today. Therefore, it is essential to ensure good QoS in a network. Indicators such as latency, jitter, and packet loss rate are considered highly relevant to QoS and user experience.<sup>(23)</sup> Different services and applications may have varying network requirements. For example, real-time communication applications and online games require low latency to ensure a smooth experience. Video streaming, with the buffering function, has a lower requirement for low latency. Real-time audio has a rather small data volume, therefore allowing only low packet loss ratio. Video on demand (VOD), by contrast, is tolerant to a higher packet loss ratio. Taken from several studies, online articles, and ITU-T standards, Table 2 shows the requirements for different network services. In this study, the latency is set to be less than 50 ms, the jitter to be lower than 30 ms, and the acceptable packet loss ratio less than 3%.

Table 2  
QoS- and user-experience-related indicators, taken from related work.

|             | Games <sup>(24–28)</sup> | Audio <sup>(29–31)</sup> | Video conference <sup>(23)</sup> |
|-------------|--------------------------|--------------------------|----------------------------------|
| Latency     | <50 ms                   | <150 ms                  | <150 ms                          |
| Jitter      | <30–50 ms                | <50 ms                   | <30 ms                           |
| Packet loss | <5%                      | <3%                      | <3%                              |

## 2.6 Differences between related work and the proposed method

Several previous studies have been dedicated to preliminary investigations on carbon-aware networks. However, to consider network stability, movements in traditional routing algorithms are normally consistent just in case the nodes move very fast and the route convergence cannot be achieved. For this reason, former methods adjust the routing path on an hourly basis and use the genetic algorithm in path planning. In addition, to maintain network stability, carbon-aware data is usually provided on the basis of the prediction model or extracted from the old databases, resulting in outdated carbon intensity data and significant errors. To deal with the above-mentioned problems and cope with AI development, we propose to use DQN for path planning. Unlike existing DQN-based routing studies that are mainly focused on QoS optimization, in our proposed method, carbon intensity, latency, jitter, and packet loss rate are jointly considered in routing decisions. In addition, together with the newly developed smart grid, frequently updated carbon intensity data may become available. Instead of making predictions, the proposed framework is designed to receive updated carbon intensity information, so that the network will be able to quickly adjust itself in accordance with the carbon intensity. For future development, smart grids should be able to output the carbon intensity of smart grid nodes in real time. After receiving the data, the network administration system can adjust the path in accordance with the carbon intensity of the grid. By integrating the QoS-related metrics, the system determines the optimal low-carbon path without compromising user experience, contributing to a carbon-aware network with a rapid routing adjustment mechanism for carbon footprint reduction. Table 3 shows the differences between the reviewed carbon-aware networking approaches and the proposed method.

## 3. Method

### 3.1 Research limitations

We aim to establish a carbon-aware routing adjustment mechanism for an AI-based smart routing protocol. Since the standards for carbon emissions in smart grids and information and communication technology are still under development, this paper is focused only on optimizing path planning for reduced carbon emissions and building a routing adjustment mechanism based

Table 3  
Characteristics of the reviewed carbon-aware studies and the proposed method.

|                              | Proposed method                                                                                       | Related work <sup>(21,22)</sup>                                                           |
|------------------------------|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Path adjustment mechanism    | Reinforcement learning DQN                                                                            | Delay-tolerant networks (DTNs) and genetic algorithm                                      |
| Carbon footprint and routing | Designed to receive frequently updated carbon intensity data and adjust routing decisions accordingly | Carbon intensity information is obtained from forecast or periodically updated data.      |
| Adjustment method            | Path selection (node switching)                                                                       | Node switching and adaptive link rate (ALR)                                               |
| QoS                          | Carbon intensity is jointly considered with latency, jitter, and packet loss rate.                    | QoS indicators are not jointly incorporated in the same manner as in the proposed method. |

on the carbon intensity. At the time of release of this paper, the mechanism has not yet been integrated with operational smart-grid systems because standardized carbon-intensity data interfaces are not currently available. In addition, the carbon intensity and QoS values used in this study are generated in a simulated environment. Therefore, the practical applicability of the proposed mechanism depends on the availability, accuracy, and update frequency of carbon intensity data provided by smart grids or energy-monitoring systems.

3.2 Objective

We aim to establish an AI-integrated carbon-aware routing protocol that considers QoS-related indicators, including CII (median life cycle carbon emissions), latency, jitter, and packet loss rate. In this way, the DQN model can devise an optimal path planning strategy that reduces carbon emissions during transmissions without sacrificing user experience.

3.3 System environment

By using the network kit to build a topology, we establish a gymnasium reinforcement learning environment and employ the DQN algorithm in a stable\_baselines3 library for training and simulation in DQNs. The system architecture is shown in Fig. 2. Table 4 shows the specifications of the hardware and environment as well as software specifications and versions.

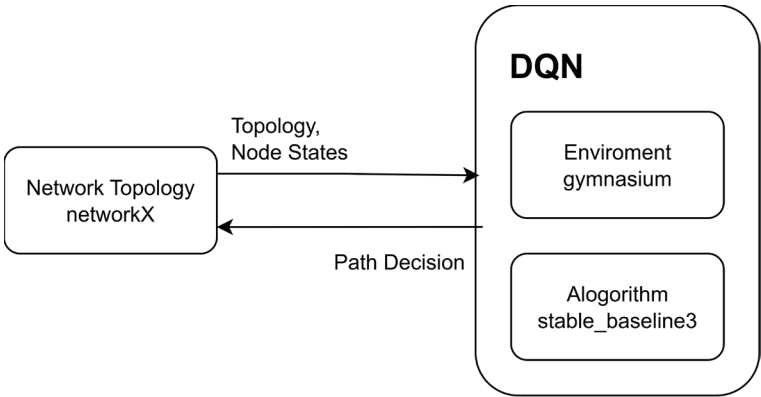


Fig. 2. System architecture.

Table 4  
System environment variables.

| Hardware and environment | Specifications and versions |
|--------------------------|-----------------------------|
| OS                       | Windows 11 Pro 23H2         |
| CPU                      | Intel Core i9-13900K 3.0GHz |
| Memory                   | 64.0 GB                     |
| GPU                      | NVIDIA RTX 4090             |
| Display memory           | 24.0 GB                     |
| Computing platform       | CUDA 12.6                   |
| Deep learning frameworks | PyTorch 2.4.1               |
| DQN environment setting  | Gymnasium 1.0.0             |
| DQN framework            | Stable-Baselines3 2.4.1     |

### 3.4 System architecture

We build a  $10 \times 5$  grid topology as the training environment; it consists of 50 nodes, each of which has horizontal, vertical, upper-right diagonal, and bottom-right diagonal connections. This topology has a total of 157 connections.

To better simulate the network instability, a single node in the model will not always remain under a good QoS status. Table 5 shows the settings adopted to generate the simulation parameters.

Figure 3 shows the process of model training. The model initializes the environment with topological data, loads the observation space for observations of metrics, loads the action space for action execution, and implements the transmission for final decision-making. If the transmission is not yet finished, the model loads the observation space again, i.e., loads the status of the next node and adjacent nodes to continue the transmission. Next, the model takes actions from the action space and continues to find the next relay node that is closer to the target. When the current node arrives at the destination, the transmission is completed. Then, the model scores this transmission. The following steps will be taken on the basis of whether or not this round is finished. If this round is not completed, the model randomly updates the node status, including latency, packet loss rate and CII, and starts all over again from the starting point and

Table 5  
Values of parameters.

| Item             | Value                                                       |
|------------------|-------------------------------------------------------------|
| Latency          | Probability of 90%: randomly generated between 0 and 1 ms   |
|                  | Probability of 10%: randomly generated between 1 and 50 ms  |
| Packet loss rate | Probability of 90%: randomly generated between 0.0 and 0.1% |
|                  | Probability of 10%: randomly generated between 0.1 and 3%   |
| CII              | Randomly generated between 11 and 820                       |

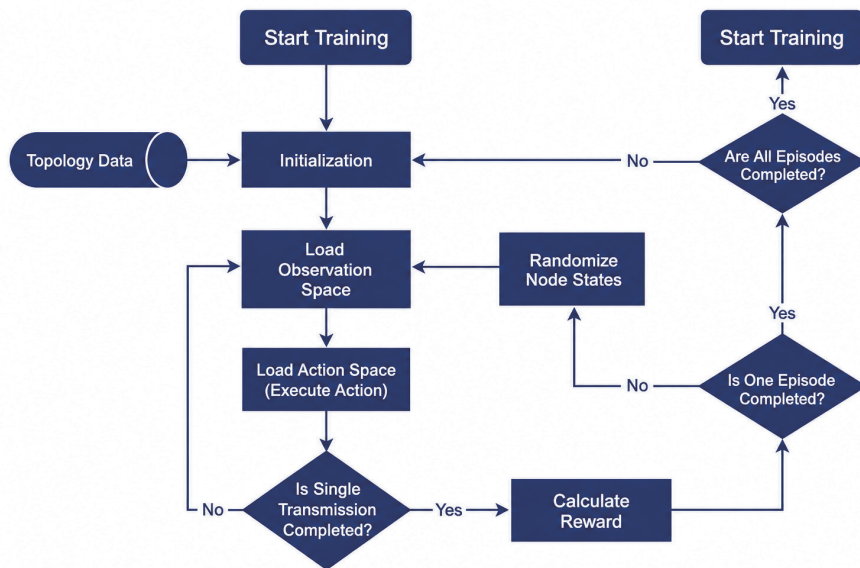


Fig. 3. (Color online) Model training.

begins the next transmission cycle. If this round is completed, the model moves forward to the next round by emptying the node's status and reinitializing the model. Such an operational process is repeated until all rounds are completed, and the model training is terminated.

The purpose of training is for the model to learn the optimal node selection strategy. For this reason, the definition of transmissions and rounds must be given so that the model computes the metrics and keeps modifying the network status, in case the model provides only a single optimal solution.

A transmission means that the model finds an optimal path from a starting point to a destination. After the transmission, the model computes the total latency and the total packet loss rate, and begins the second transmission. As the second transmission is completed, the model computes the second total latency and calculates the difference between the two total latencies, namely, jitter. In a single transmission, a node does not change index parameters to match real transmission states. One round consists of 100 transmissions. Once a round is finished, the environment will be reset for proper variations.

Figure 4 shows that a single transmission starts from “loading the observation space” and progresses to “judging if the transmission is terminated”. As the transmission starts, the model makes calculations in accordance with the status of transmission and space, checks the available valid action space, and loads the action space for action execution. To prevent network loops, a node that has been visited is set to be invalid in case the model revisits visited nodes, which would result in network loops. If the model cannot find valid actions in the action space, the transmission fails. If the action space contains valid actions, the model chooses the best action

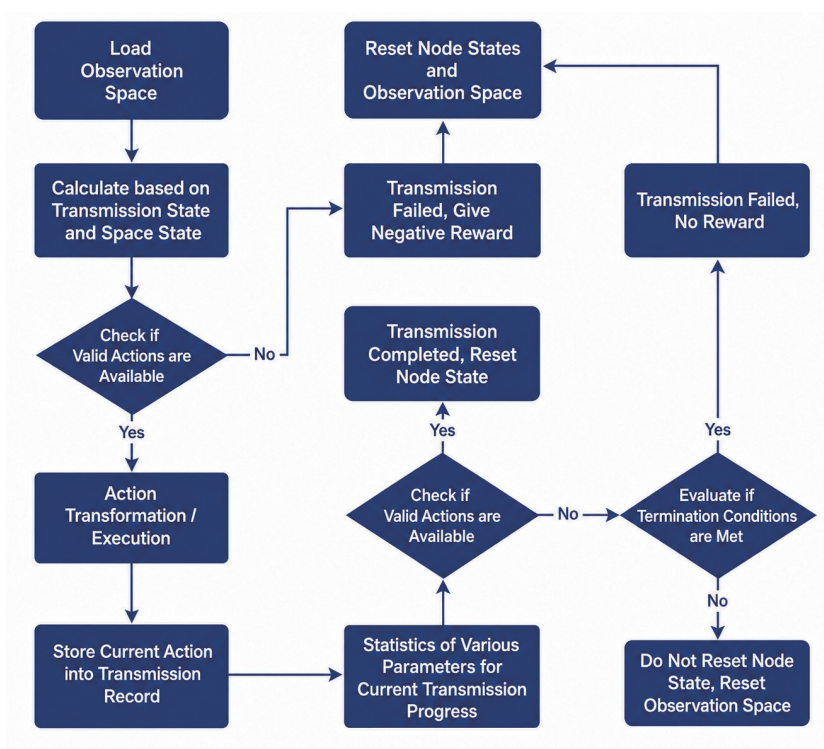


Fig. 4. (Color online) Flowchart of a single transmission.

on the basis of the selection strategy, changes actions, and switches to a new status. Once switched to the new status, the model keeps accumulating the metrics and judges whether the destination is reached. On detecting the destination, the model decides that the transmission is completed, calculates the reward scores, and resets the node status. If the destination is not detected, the model computes the metrics accumulated from the starting point to the current node to judge if the network has exceeded the lowest standard. If not, the model maintains the same environment and continues the transmission. If the lowest standard is exceeded, the transmission is judged to have failed, and the node status is updated for the next transmission.

During the transmission, the model calculates the reward scores following the flowchart, and these scores can affect the strategy for choosing the next action. After several repetitions of training, the model discovers a strategy with higher scores and learns the routing adjustment mechanism.

As for the reward function, the model, in the current learning strategy, chooses the best action in the action space. If the model cannot find valid space, the transmission fails and a score of minus 1000 is rewarded to the model. If the transmission is successful, a score of 1000 is rewarded to the model; scores regarding QoS and CII are also given in accordance with the quality of the transmission. Table 6 shows the method of calculating reward scores for each metric, and the reward scores are explained in Table 7.

Table 6  
Method of calculating reward scores.

|                                                                                                   |     |
|---------------------------------------------------------------------------------------------------|-----|
| $R_{Latency} = \left(1 - \min\left(1, \frac{L_{total}}{50}\right)\right) \times 200$              | (4) |
| $R_{Packetloss} = \left(1 - \min\left(1, \frac{P_{total}}{0.03}\right)\right) \times 200$         | (5) |
| $R_{Jitter} = \left(1 - \min\left(1, \frac{J}{30}\right)\right) \times 200$                       | (6) |
| $R_{CII} = \left(1 - \frac{CII_{total} - CII_{best}}{CII_{worst} - CII_{best}}\right) \times 400$ | (7) |

Table 7  
Reward scores.

| Transmission status | State                                                         | Score                                                                                                                                                                                                                                                                               |
|---------------------|---------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                     | No valid action                                               | −1000                                                                                                                                                                                                                                                                               |
| Failure             | QoS exceeds the standard (early termination of transmission). | 0                                                                                                                                                                                                                                                                                   |
| Success             | QoS matches the standard.                                     | <p>In a successful transmission, the model receives a basic score as well as an additional reward score in accordance with the QoS metrics.</p> <p>The total score R is</p> $R = 1000 + R_{Latency} + R_{Jitter} + R_{Packetloss} + R_{CII},$ <p>ranging between 1000 and 2000.</p> |

### 3.5 DQN model parameters

We set the values of the DQN hyperparameters as listed in Table 8. Next, we introduce the architecture of the CNN model and describe how the CNN model loads data from the observation space to the convolutional layer of the model and how the final outcome is output.

Converting the observation space: The model defines four channels of the observation space—latency, packet loss rate, CII, and node status (including current status, target status, and visited nodes)—as the “form” (4, height, width) and inputs the data into the convolutional layer. Figure 5 shows the code of the observation space.

Figure 6 shows the code of the CNN convolutional layer. It was designed as follows.

First convolutional layer: Four input channels are convolved with  $32 \ 3 \times 3$  filters. Together with padding = 1 and stride = 1, the output feature map remains the same in height and width. In the meantime, the output is 32-dimensional and passes through the activation function ReLu.

Table 8  
DQN hyperparameters.

| Parameter               | Value              |
|-------------------------|--------------------|
| policy                  | CnnPolicy          |
| learning_rate           | $5 \times 10^{-4}$ |
| batch_size              | 256                |
| buffer_size             | 200000             |
| learning_starts         | 200000             |
| exploration_fraction    | 0.9                |
| exploration_initial_eps | 1                  |
| exploration_final_eps   | 0.1                |
| target_update_interval  | 1000               |
| train_freq              | 1                  |
| gradient_steps          | 2                  |
| total_timesteps         | 4000000            |
| max_grad_norm           | 10.0               |
| double_q                | True               |

```
super(CNN, self).__init__(observation_space, features_dim)
n_input_channels = observation_space.shape[0]
```

Fig. 5. (Color online) Observation space denoted in code.

```
self.cnn = nn.Sequential(
    nn.Conv2d(n_input_channels, 32, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.Flatten()
)
```

Fig. 6. (Color online) Code of CNN convolutional layer.

Second convolutional layer:  $64 \ 3 \times 3$  filters are used to process the 32 channel outputs of the previous layer. The spatial dimensionality basically remains the same in height and width. Therefore, the output shape is (64, height, width) and goes through the activation function ReLu.

Third convolutional layer: The number of channels remains 64 in this layer, but one more  $3 \times 3$  convolution is triggered to learn the hidden features, and it also goes through the activation function ReLu.

Flatten: The output of the 3D convolution (64, height, width) is flattened into a vector of  $64 \times \text{height} \times \text{width}$  size.

With the flattened layer and an additional linear layer, the high-dimensional feature is converted to features\_dim. The sum of CNN + Flatten + Linear will be output into a vector with shape = (batch\_size, features\_dim). Figure 7 shows the source code. The generated feature vector is sent back to the DNN to compute the Q-value and policy logits at the backend.

## 4. Experiments and Results

This study conducts two experiments to evaluate the trained model. The result is explained below.

### 4.1 Experiment 1: Comparing all metrics

In Experiment 1, the QoS and CII metrics are compared between the Dijkstra algorithm, which only searches for the shortest path, and our proposed carbon-aware DQN method, which adjusts the routing strategy on the basis of QoS and CII.

Assume that QoS is good in the simulation, i.e., 90% of nodes maintain stable latency and package loss rate, and the QoS and CII metrics in every transmission from node a2 to node i4 are recorded. Figure 8 shows the network topology and node labels used in Experiment 1, and Table 9 shows the values of the experiment parameters.

During the simulation, each node randomly changes the QoS and CII metrics in accordance with the above-listed probability in 100 transmissions, and the carbon-aware DQN routing mechanism searches for the optimal transmission path on the basis of the node status. In each transmission, the method computes the status of all the visited nodes in the transmission, calculates the QoS and CII of the DQN routing mechanism, and compares the result with Dijkstra's algorithm that finds the shortest path. Figure 9 shows the result of 100 transmissions.

```
self.linear = nn.Sequential(
    nn.Linear(n_flatten, features_dim),
    nn.ReLU()
)
```

Fig. 7. (Color online) Conversion of high-dimensional feature into single vector code.

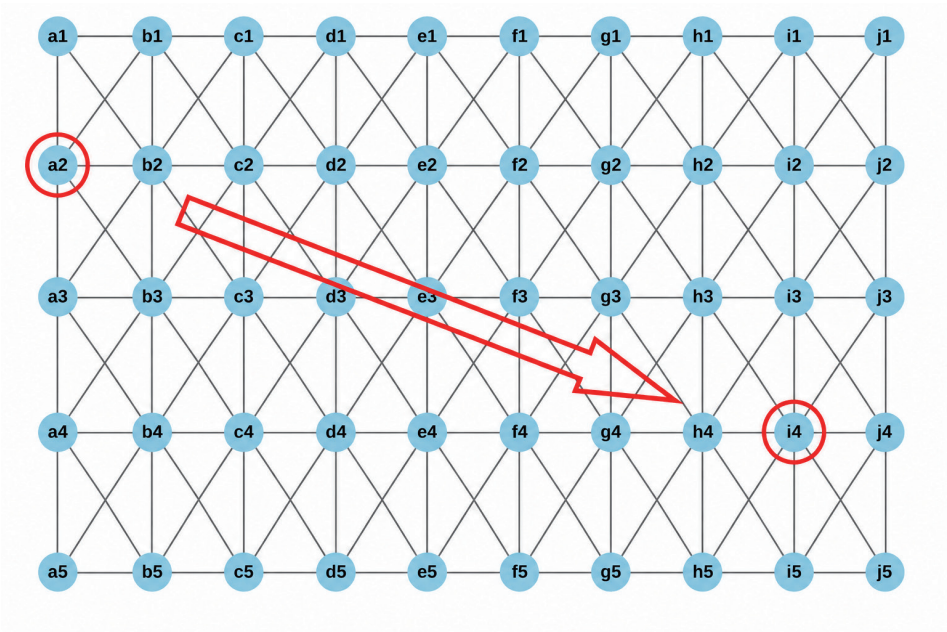


Fig. 8. (Color online) Topology and nodes.

Table 9  
Experiment 1: Settings of the 90% stable nodes in the topology.

| Item         | Value                                                       |
|--------------|-------------------------------------------------------------|
| Latency      | Probability of 90%: Randomly generated between 0 and 1 ms   |
|              | Probability of 10%: Randomly generated between 1 and 50 ms  |
| Package loss | Probability of 90%: Randomly generated between 0.0 and 0.1% |
|              | Probability of 10%: Randomly generated between 0.1 and 3%   |
| CII          | Randomly generated between 11 and 820                       |

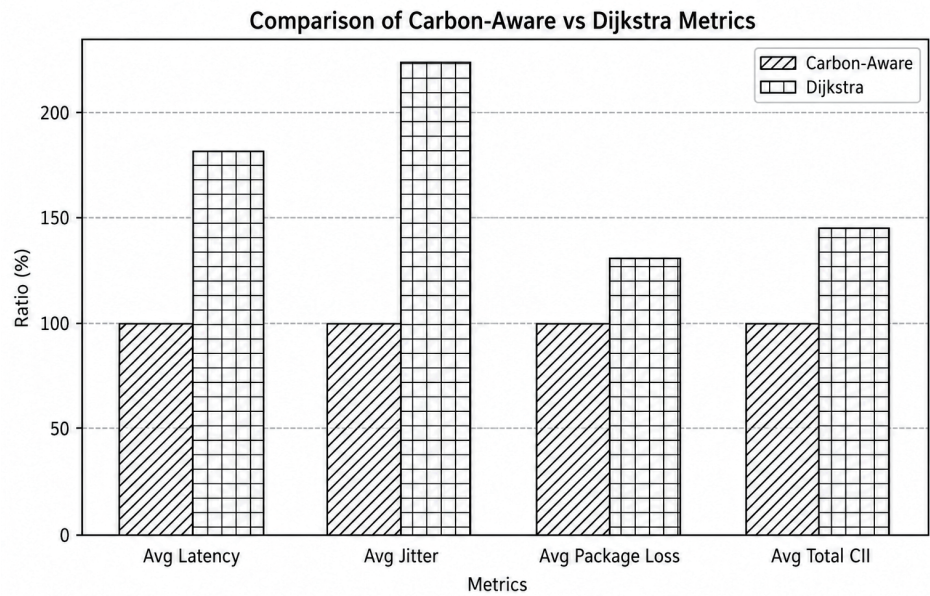


Fig. 9. Experiment 1: Carbon-aware vs Dijkstra metrics.

Figure 9 reveals that the proposed carbon-aware method obviously outperforms Dijkstra's algorithm: 45.2% less latency than that of Dijkstra, 55.9% less jitter than in the case of Dijkstra, and 25% less package loss than that of Dijkstra. In other words, the path selected by the carbon-aware method better stabilizes QoS, and carbon intensity can be reduced by 35%. This reveals that as far as user experience and carbon reduction are concerned, the routing mechanism that considers QoS and carbon intensity performs better than traditional routing protocols.

#### 4.2 Experiment 2: Inverse ratio between QoS and carbon intensity

Experiment 2 reveals the model's capability to adjust QoS and carbon intensity. The same topology is used in Experiment 2, but the node status is distributed in line with the following rule. In terms of carbon intensity, edge nodes are designed for low-carbon operation while other nodes have high carbon intensity.

The calculation below is performed to calculate the status of edge nodes. First, calculate the maximum distance within the topological space.

$$D_{max} = \sqrt{(9 - 4.5)^2 + (4 - 2)^2} \quad (8)$$

Next, calculate the distance between the node and the center. This is a  $10 \times 5$  topology and the topological center is located at (4.5, 2). Thus, the formula is

$$D = \sqrt{(col - 4.5)^2 - (row - 2)^2}. \quad (9)$$

After standardizing the two values, calculate if the node is on the edge.

$$D_{norm} = \frac{D}{D_{max}} \quad (10)$$

Also, check whether the node with the value  $D_{norm}$  is exactly on the edge. In the experiment, the threshold value of  $D_{norm}$  is set to 0.7. Next, nodes are divided in accordance with the maximum distance to the center in the portion 7:3. Nodes with a difference within 0.7 to the standardized value are defined as central nodes, and the rest are edge nodes. Table 10 presents the settings used to configure the node states.

In this experiment, the central and edge nodes are separated, and there are different strategies to generate node metrics. Central nodes maintain better QoS and higher carbon intensity, while edge nodes have lower carbon intensity, but QoS could be decreased with an increasing number of transmissions. Such a topology is designed to evaluate whether the model can achieve the envisioned goal. At the beginning, the model selects the edge nodes with low carbon intensity as well as low latency, jitter, and package loss. With the increasing number of transmissions, QoS becomes worse than even the minimum QoS standard in the experiment and the model begins to

select the central nodes with better QoS but higher carbon intensity. The results obtained after 100 transmissions are completed with the topology settings listed in Table 10 are shown in Figs. 10 to 12.

Figures 10, 11, and 12 show the variations in latency, jitter, and packet loss, respectively, and carbon intensity. When a node starts a transmission, the values of the above-mentioned metrics are low, indicating good QoS, and the CII value falls to around 250. The increasing number of transmissions leads to QoS degradation, and the QoS status gradually becomes poor as it approaches the 50th transmission. Since nodes with low carbon intensity will no longer satisfy

Table 10  
Experiment 2: Generating the metric settings.

| Node         | Indicator    | Range of random parameter values                                                  |
|--------------|--------------|-----------------------------------------------------------------------------------|
| Central node | Latency      | Probability of 99%: 0–1 ms                                                        |
|              |              | Probability of 1%: 1–50 ms                                                        |
|              | Package loss | Probability of 99%: 0.0–0.1%                                                      |
|              |              | Probability of 1%: 0.1–3%                                                         |
| Edge node    | CII          | 24–820                                                                            |
|              | Latency      | First transmission:<br>Probability of 100%: 0–1 ms                                |
|              |              | 100th transmission:<br>Probability of 30%: 0–1 ms                                 |
|              |              | Probability of 70%: 1–50 ms                                                       |
|              |              | First transmission:<br>Probability of 100%: 0.0–0.1%                              |
|              | Package loss | 100th transmission:<br>Probability of 30%:<br>0.0–0.1% Probability of 70%: 0.1–3% |
|              |              | CII 11–24                                                                         |

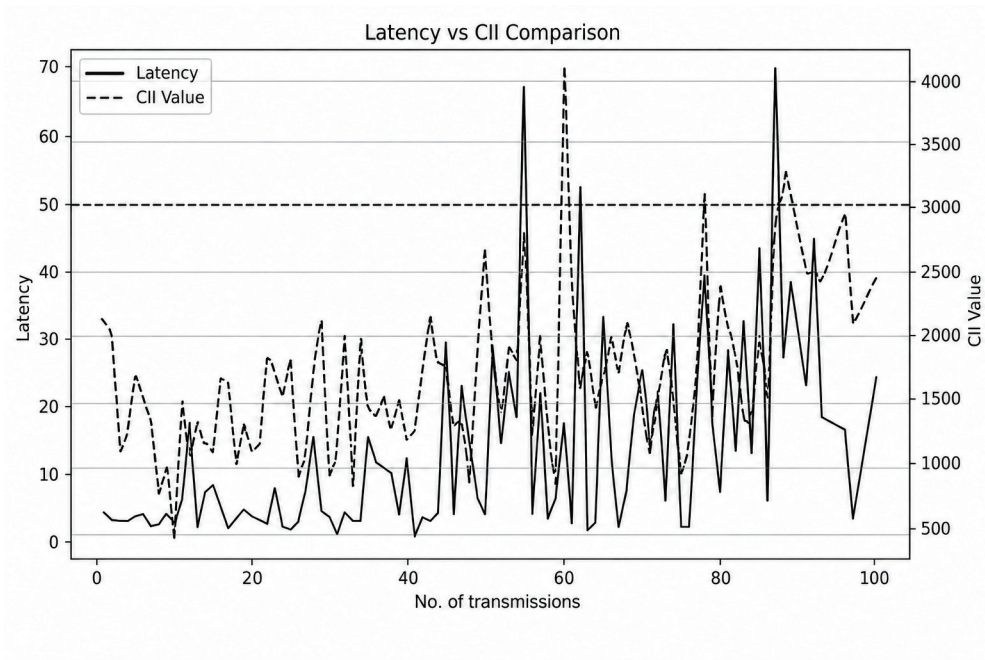


Fig. 10. Experiment 2: Latency vs CII.

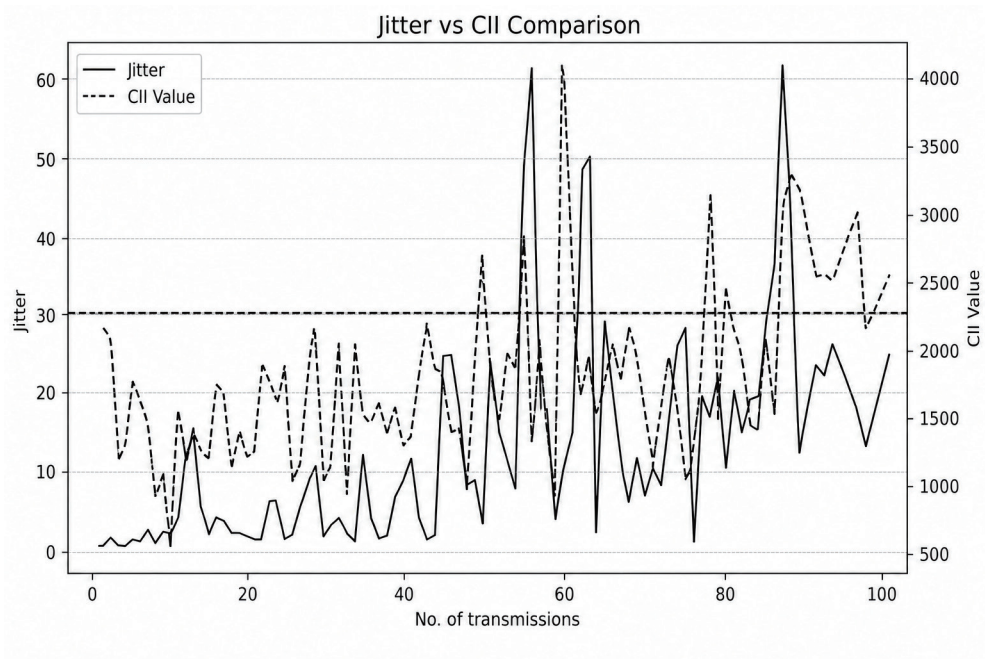


Fig. 11. Experiment 2: Jitter vs CII.

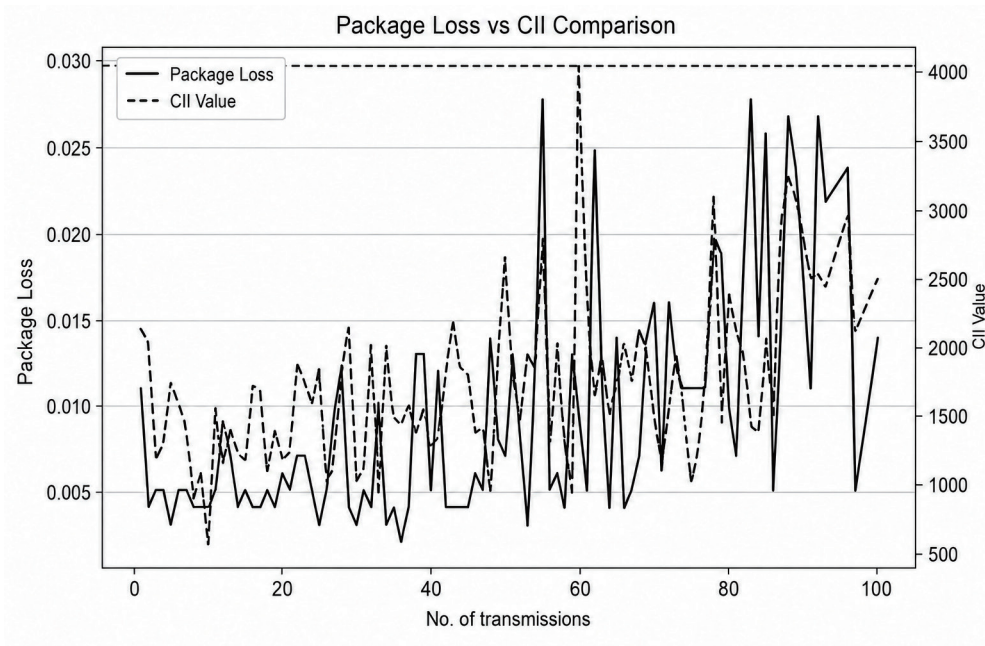


Fig. 12. Experiment 2: Package loss vs CII.

QoS needs, the routing adjustment mechanism begins to search for nodes with higher carbon intensity to maintain good QoS. It can be seen that the carbon intensity is increased to 2500 to achieve a QoS status no worse than the worst standard, showing the method’s ability to take both QoS and carbon intensity into account.

## 5. Conclusions

We proposed a DQN-based routing adjustment mechanism that uses DQN to find the optimal strategy for achieving the best path planning performance corresponding to the carbon intensity of nodes with various power sources. Future smart grid development will enable grids to output carbon intensity. With the information, nodes can use the proposed method for path planning, find low-carbon routing paths, and ensure that QoS can be maintained without sacrificing user experience. In this way, a low-carbon smart routing method that considers both QoS and carbon intensity can be achieved. Low-carbon energy sources such as solar and wind power that are easily affected by the environment will affect the proportion of energy consumption in different time intervals and lead to frequent variations in node carbon intensity. Our proposed routing adjustment mechanism is capable of coping with such high variability to avoid QoS instability caused by traditional algorithms that account for carbon intensity and wait for path convergence. However, the current results are based on a simulated network environment. The real-world performance of the proposed mechanism may depend on the availability, accuracy, and update frequency of carbon intensity and network status data. In future work, therefore, we will evaluate the mechanism using operational network measurements and carbon-intensity data obtained from smart-grid or energy-monitoring systems.

## References

- 1 C. Westphal and A. Clemm: Proc. 2023 Int. Conf. Computing, Networking and Communications (ICNC) (IEEE, 2023) 581–585. <https://doi.org/10.1109/ICNC57223.2023.10074563>
- 2 T. Bruckner, I. A. Bashmakov, Y. Mulugetta, H. Chum, A. de la Vega Navarro, J. Edmonds, A. Faaij, B. Fungtammasan, A. Garg, E. Hertwich, D. Honnery, D. Infield, M. Kainuma, S. Khennas, S. Kim, H. B. Nimir, K. Riahi, N. Strachan, R. Wisser, and X. Zhang: Energy Systems, in Climate Change 2014: Mitigation of Climate Change. Contribution of Working Group III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change (Cambridge University Press, Cambridge, 2014), Chap. 7. [https://www.ipcc.ch/site/assets/uploads/2018/02/ipcc\\_wg3\\_ar5\\_chapter7.pdf](https://www.ipcc.ch/site/assets/uploads/2018/02/ipcc_wg3_ar5_chapter7.pdf)
- 3 Cisco: Cisco Annual Internet Report (2018-2023) White Paper (Cisco, 2020). <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>
- 4 J. Lorincz, A. Capone, and J. Wu: Sensors **19** (2019) 4864. <https://doi.org/10.3390/s19224864>
- 5 E. W. Dijkstra: A Note on Two Problems in Connexion with Graphs, in Edsger Wybe Dijkstra: His Life, Work, and Legacy (ACM, New York, 2022) pp. 287–290. <https://doi.org/10.1145/3544585.3544600>
- 6 Q. Zheng, J. Ma, Q. Liu, and P. Liang: Proc. 2024 Int. Conf. Networking and Network Applications (NaNA) (IEEE, 2024) 258–262.
- 7 S. Ye, L. Xu, Z. Xu, and F. Wang: Alexandria Eng. J. **107** (2024) 317. <https://doi.org/10.1016/j.aej.2024.07.045>
- 8 H. Tabrizi, G. Farhadi, and J. M. Cioffi: Proc. 2012 IEEE Int. Conf. Communications (ICC) (IEEE, 2012) 3217–3222. <https://doi.org/10.1109/ICC.2012.6364194>
- 9 S. He, T. Wang, and S. Wang: Proc. GLOBECOM 2020 IEEE Global Communications Conf. (IEEE, 2020) 1–6. <https://doi.org/10.1109/GLOBECOM42002.2020.9322449>
- 10 D.-N. Xiong and Y. Li: J. Commun. **36** (2015) 252. <https://doi.org/10.11959/j.issn.1000-436x.2015240>
- 11 H. Xu, D. Li, M. Liu, G. Han, W. Huang, and C. Xu: IEEE Trans. Veh. Technol. **69** (2020) 10127. <https://doi.org/10.1109/TVT.2020.3000908>
- 12 Q. Fu, E. Sun, K. Meng, M. Li, and Y. Zhang: IEEE Access **8** (2020) 103491. <https://doi.org/10.1109/ACCESS.2020.2995511>
- 13 C. Liu, M. Xu, Y. Yang, and N. Geng: Proc. IEEE INFOCOM 2021 IEEE Conf. Computer Communications (IEEE, 2021) 1–10. <https://doi.org/10.1109/INFOCOM42981.2021.9488736>
- 14 A. A. Magadam, A. Ranjan, and D. G. Narayan: Proc. 2021 12th Int. Conf. Computing Communication and Networking Technologies (ICCCNT) (IEEE, 2021) 1–7. <https://doi.org/10.1109/ICCCNT51525.2021.9579514>

- 15 R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, (MIT Press, Cambridge, MA, 2018) 2nd ed. <https://mitpress.mit.edu/9780262039246/reinforcement-learning/>
- 16 Y. Kong, Y. Li, and C.-H. Hsia: J. Internet Technol. **25** (2024) 955. <https://jit.ndhu.edu.tw/article/view/3121>
- 17 Z. Yang, Y. Kong, and C.-H. Hsia: J. Internet Technol. **25** (2024) 79. <https://jit.ndhu.edu.tw/article/view/3014>
- 18 Y. Kong, J. Wei, and C.-H. Hsia: J. Internet Technol. **26** (2025) 293. <https://jit.ndhu.edu.tw/article/view/3168>
- 19 A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, and W. Cirne: IEEE Trans. Power Syst. **38** (2023) 1270. <https://doi.org/10.1109/TPWRS.2022.3173250>
- 20 N. Zilberman, E. M. Schooler, U. Cummings, R. Manohar, D. Nafus, R. Soulé, and R. Taylor: ACM SIGENERGY Energy Inf. Rev. **3** (2023) 15. <https://doi.org/10.1145/3630614.3630618>
- 21 Md. M. Hossain, J.-P. Georges, E. Rondeau, and T. Divoux: IFAC-PapersOnLine **53** (2020) 8303. <https://doi.org/10.1016/j.ifacol.2020.12.1905>
- 22 S. Tabaeiaghdaei, S. Scherrer, J. Kwon, and A. Perrig: Proc. 14th ACM Int. Conf. Future Energy Systems (e-Energy) (ACM, New York, 2023) 144–158. <https://doi.org/10.1145/3575813.3595192>
- 23 S. Duchin: What Is Packet Loss, What Causes It and How to Reduce It? <https://www.teridion.com/blog/network-performance/how-to-fix-packet-loss/> (accessed October 2026).
- 24 V. Courtemanche and A. Desveaux: The Effects of Latency in Commercial Cloud Video Gaming Services (Worcester Polytechnic Institute, Worcester, MA, 2020). <https://web.cs.wpi.edu/~claypool/iqp/cloudgame-19/report.pdf>
- 25 S. S. Sabet, S. Schmidt, C. Griwodz, and S. Möller: Proc. 2019 Eleventh Int. Conf. Quality of Multimedia Experience (QoMEX) (IEEE, 2019) 1–6. <https://doi.org/10.1109/QoMEX.2019.8743239>
- 26 J. P. McManus, T. G. Day, and Z. J. Mailloux: The Effects of Latency, Bandwidth, and Packet Loss on Cloud-Based Gaming Services (Worcester Polytechnic Institute, Worcester, MA, 2019).
- 27 S. Vlahovic, M. Suznjec, and L. Skorin-Kapov: Proc. 2019 Eleventh Int. Conf. Quality of Multimedia Experience (QoMEX) (IEEE, 2019) 1–3. <https://doi.org/10.1109/QoMEX.2019.8743193>
- 28 S. S. Sabet: The Influence of Delay on Cloud Gaming Quality of Experience (Springer, Cham, 2023). <https://doi.org/10.1007/978-3-030-99869-1>
- 29 Digital Samba: Jitter Explained: Causes, Effects, and Mitigation in Networking (2024). [https://medium.com/@digital\\_samba/jitter-explained-causes-effects-and-mitigation-in-networking-cfa200342a5d](https://medium.com/@digital_samba/jitter-explained-causes-effects-and-mitigation-in-networking-cfa200342a5d) (accessed October 2026).
- 30 ITU-T: One-Way Transmission Time, Recommendation G.114 (International Telecommunication Union, Geneva, 2003). <https://www.itu.int/rec/T-REC-G.114-200305-I/en>
- 31 IR: Network Jitter - Common Causes and Best Solutions. <https://www.ir.com/guides/what-is-network-jitter> (accessed October 2026).