Future Technology
Open Access Journal
ISSN 2832-0379

November 2026| Volume 05 | Issue 04 | Pages 74-85

Journal homepage: https://fupubco.com/futech

https://doi.org/10.55670/fpll.futech.5.4.7

Future Publishing LLC

Article

Deep Q-network path planning with stratified sampling and prioritized replay

Yuxing Zhang, Setyawan Widyarto*

Faculty of Communication, Visual Art & Computing, Universiti Selangor,45600 Bestari Jaya, Selangor, Malaysia

ARTICLE INFO

Article history:
Received 14 February 2026
Received in revised form
16 June 2026
Accepted 21 July 2026

Keywords:
Deep Q-network, Stratified sampling,
Prioritized experience replay,
Mobile robot path planning,
Deep reinforcement learning

*Corresponding author
Email address:
9212000501@student.unisel.edu.my

DOI: 10.55670/fpll.futech.5.4.7

ABSTRACT

Deep reinforcement learning (DRL) has been widely adopted for mobile robot path planning in unknown and complex environments. However, existing DRL-based approaches suffer from two main limitations: sparse reward, which provides meaningful feedback only at terminal states, and experience heterogeneity, which reduces the efficiency of uniform sampling from the standard experience replay buffer due to the diversity of transition qualities stored within it. To tackle both issues at once, we propose SS-PER-DQN, a novel path-planning approach in which stratified sampling and a prioritized experience replay mechanism are incorporated into the Double Deep Q-Network. The memory buffer is divided into three priority layers based on temporal-difference error levels, with a dynamically varying sampling ratio between layers, and importance-sampling compensation to balance the sampling-distribution bias. A densely shaped reward function is additionally employed to expedite early exploration in the presence of sparsely rewarded actions in the environment. Experiments performed on Grid World and ROS Gazebo simulations show that the SS-PER-DQN is able to successfully plan paths at success rates of 94.5% and 89.0%, with a 24.5% and 21.8% reduction in path lengths, respectively, when compared to the traditional DQN algorithm, after convergence in approximately 53% fewer training episodes. The analysis conducted during the ablation study reveals the specific roles of each contribution: stratified sampling helps the algorithm converge early by ensuring that informative state transitions guide the first phase of learning, and importance weighting influences the quality of gradient updates in later stages. It is also observed that the absolute planning time per decision step ranges from 5.0 ms (Grid World) to 5.3 ms (Gazebo) for SS-PER-DQN, representing an incremental overhead of approximately 1.9–2.2 ms per step relative to the classic DQN baseline, which remains well within the real-time requirements for robot navigation.

1. Introduction

Mobile robot path planning has become a critical issue in intelligent robotics, where the robot needs to move from a start point to an end point without colliding with obstacles in a complex environment [1]. The rise of artificial intelligence has spurred deep reinforcement learning (DRL), which can help robots develop policies to navigate their surroundings without relying on environmental maps [2]. In the context of DRL algorithms, Mnih et al.'s Deep Q-Network (DQN) algorithm has shown remarkable results in making decisions under uncertainty by utilizing Q-learning and deep neural networks [3,4], where Ref [3] refers to the original arXiv preprint and Ref [4] to the peer-reviewed Nature publication of the same work [3]; however, the traditional design of the DQN framework faces some difficulties in converging quickly due to Q-value overestimation, hence motivating a new generation of enhanced versions of DQN [5]. Improved versions like Double DQN and Dueling DQN have gained popularity in mobile robot navigation problems by mitigating overestimation and value-function decomposition [6], and several studies have implemented these frameworks by integrating them with problem-specific techniques [7-9]. One significant drawback of implementing DRL for path planning tasks is the inefficiency of experience replay. By uniformly sampling from all previously learned transitions, traditional approaches result in the repetition of low-learning experiences while neglecting the most informative ones. This limitation inspired researchers to introduce prioritized experience replay (PER), which gives a higher probability to sampling from experiences with large TD error values, so that learning concentrates only on the most useful examples [10]. Based on this premise, a number of experiments have

confirmed that incorporating PER into the DQN-based path-planning task leads to much faster convergence and higher success rates across various complex grid worlds and Gazebo simulation platforms [11-13]. However, one issue with using PER is that, due to its reliance on a single scalar priority, it is prone to sampling bias when experiences in the replay memory pool exhibit high heterogeneity [14].

In addition to sampling efficiency, the sparse reward challenge remains a recurring issue in DRL-based path planning, in which agents receive reward feedback only when they reach the destination or bump into an obstacle, making most other states irrelevant and rendering training inefficient [15,16]. HER resolves this issue by retroactively relabeling rewards for failed episodes, creating dense synthetic rewards from negative experiences [17]. Multi-step HER and curriculum-based HER have been used to enhance sampling efficiency but suffer from off-policy bias with multi-step returns, making them unsuitable for long-horizon tasks [18]. There have been several recent applications of HER with actor-critic architectures for robotic arm manipulation and mobile robot navigation, where considerable improvement in convergence rates was observed [19,20]; however, the implementation of a structured sampling hierarchy and prioritized experience replay for HER within the DQN framework for ground-robot navigation has yet to be attempted. Recent efforts to address these limitations have explored complementary strategies, including attention-based prioritization combined with PER [21] and heuristic reward shaping with dynamic windowing [22]. Despite these advances, key weaknesses persist in existing DRL-based navigation methods: inefficient exploitation of valuable experiences, slow convergence under sparse rewards, and limited adaptability across environment types [1,5,6]. Stratified decomposition of the replay buffer offers a principled approach to reconciling prioritization with sampling efficiency [2].

To overcome these limitations, this paper presents a new path-planning approach, SS-PER-DQN, where SS denotes Stratified Sampling, and PER denotes Prioritized Experience Replay. This approach combines stratified sampling (SS) together with the prioritized experience replay (PER) technique within the framework of the Double DQN. In particular, the replay memory is divided into three groups, i.e., high, medium, and low priorities, according to TD error thresholds. Then, during learning, a dynamic inter-layer sampling rate is introduced to ensure that useful transitions are sampled more often while still maintaining sufficient diversity. An importance sampling technique can be used to adjust the skewed distribution. Furthermore, this paper provides an improved reward function for sparse-reward grids/Gazebo environments to facilitate early exploration. Through simulations, it can be verified that the proposed approach converges faster and plans paths with greater accuracy and shorter lengths than the basic DQN, PER-DQN, and baseline Subsampling DQN approaches, respectively.

The objectives of this work are as follows: (1) to propose SS-PER-DQN, a novel path planning framework that integrates stratified experience replay buffer decomposition with prioritized sampling and importance sampling correction into the Double Deep Q-Network architecture; (2) to design a dense reward shaping function that alleviates the sparse reward problem in grid-based and continuous robot navigation environments, thereby accelerating early-stage policy learning; (3) to validate the effectiveness and efficiency of the proposed approach through systematic experiments in both discrete Grid World and continuous ROS Gazebo simulation environments, benchmarking against classic DQN, PER-DQN, and SS-DQN baselines; and (4) to quantify the individual contributions of stratified sampling and prioritized experience replay through ablation studies, establishing the complementary roles of each component in improving convergence speed and navigation performance.

2. Proposed method

2.1 State space and reward function design

The proposed solution uses the Kinect depth camera as the primary sensing modality, allowing the robot to perceive its surroundings without relying on any prebuilt map information or GPS. It should be noted that the sensing modality and state representation differ between the two experimental environments used in this study. In the Grid World environment, a Kinect depth camera serves as the primary sensor, and the depth image is processed through a convolutional neural network (CNN) to extract a flattened feature vector, which is then concatenated with the robot's current position coordinates (x_t, y_t), heading angle θ_t, and relative displacement to the target $(\Delta x_t, \Delta y_t)$ to form the full state S_t. In the Gazebo simulation environment, a laser range finder is used instead, producing a set of distance readings that are directly concatenated with the robot's pose and goal position information to form a ten-dimensional continuous state vector. Accordingly, the network architecture adopted in each environment differs: a CNN-based architecture is used for the Grid World setting to process the high-dimensional depth-image input, whereas a multi-layer perceptron (MLP) is employed in the Gazebo setting to process the low-dimensional laser-based state vector. The SS-PER-DQN framework accommodates both configurations by keeping the stratified buffer structure, dynamic sampling ratios, and importance-sampling correction independent of the specific network architecture, so that only the input layer and feature-extraction module need to be adapted across environments. At each time step t, the environmental state S_t is represented as a concatenation of the depth image feature vector extracted from the Kinect sensor, the robot's current position coordinates (x_t, y_t), its heading angle θ_t, and the relative displacement vector to the target point $(\Delta x_t, \Delta y_t)$, formally defined as $s_t = [f_{\text{depth}}, x_t, y_t, \theta_t, \Delta x_t, \Delta y_t]$, where $f_{\text{depth}} \in \mathbb{R}^d$ denotes the flattened depth feature representation. Action space A is defined as a discrete set of motion primitives comprising forward movement and angular turns at fixed increments, enabling the agent to execute deterministic navigation commands at each decision step.

A central challenge in DRL-based path planning is the sparsity of environmental rewards, wherein the agent receives meaningful feedback only at terminal states. To mitigate this, we design a shaped reward function r_t that provides dense intermediate signals throughout each episode. The reward function is formally defined as:

$$r_t = \begin{cases} r_{\text{goal}} \text{ if } d_t \le d_{\text{threshold}} \\ r_{\text{collision}} \text{ if obstacle detected} \\ \lambda_1(d_{t-1} - d_t) - \\ \lambda_2 \cdot 1[\Delta\theta_t > \theta \text{step otherwise}_{max} \end{cases} \quad (1)$$

where d_t denotes the Euclidean distance between the robot and the goal at time step t, $r_{\text{goal}} = +10$ is the terminal success reward, $r_{\text{collision}} = -10$ is the penalty for obstacle contact, λ_1 is a positive scaling coefficient for distance-based progress, λ_2 penalizes excessive angular deviation, and is a small constant step penalty that discourages unnecessary wandering. The term $\lambda_1(d_{t-1} - d_t)$ provides continuous positive feedback

when the robot moves closer to the goal and negative feedback when it drifts away, effectively transforming the sparse goal-reaching problem into a dense reward landscape that substantially accelerates early-stage policy learning. Specifically, in all experiments, the reward parameters are set as follows: r_{goal}=+10, $r_{collision}$=-10, $\lambda_1 = 1.0$, $\lambda_1 = 0.5$, $\theta_{max} = 30^0$, and C_{step}=0.01. These values were determined empirically through preliminary tuning to balance goal-reaching incentives, obstacle avoidance penalties, and step efficiency.

2.2 Stratified sampling mechanism

The core innovation of the proposed method lies in decomposing the experience replay buffer D into three priority tiers based on the magnitude of the temporal difference (TD) error associated with each stored transition. For a transition $e_i = (s_i, a_i, r_i, s_i')$, the TD error is computed as $\delta_i = |r_i + \gamma \, max_{a'} Q(s_i', a'; \theta^-) - Q(s_i, a_i; \theta)|$, where θ and θ^- denote the parameters of the online and target networks respectively, and γ is the discount factor. The buffer is partitioned into a high-priority tier D_H, a medium-priority tier $\mathcal{D}_M$, and a low-priority tier $\mathcal{D}_L$ according to TD error thresholds τ_H and τ_L, such that $\mathcal{D}_H = \{e_i \mid \delta_i \geq \tau_H\}$, $\mathcal{D}_M = \{e_i \mid \tau_L \leq \delta_i < \tau_H\}$, and $\mathcal{D}_L = \{e_i \mid \delta_i < \tau_L\}$. The thresholds δ_1 = 0.6 and δ_2 = 0.2 were determined through preliminary grid search experiments, where values in the range [0.1, 1.0] were evaluated; the selected values yielded the most stable tier distribution across training episodes.

Rather than applying fixed inter-layer sampling ratios, the proposed mechanism dynamically adjusts the proportion of samples drawn from each tier across training iterations. Let $\rho_H^{(k)}$, $\rho_M^{(k)}$, and $\rho_L^{(k)}$ denote the sampling ratios at the training step k, satisfying $\rho_H^{(k)} + \rho_M^{(k)} + \rho_L^{(k)} = 1$. The ratios are updated according to the mean TD error of each tier at the preceding step:

$$\rho_j^{(k)} = \frac{\bar{\delta}_j^{(k-1)}}{\sum_{j \in \{H,M,L\}} \bar{\delta}_j^{(k-1)}}, \quad j \in \{H, M, L\} \tag{2}$$

where $\bar{\delta}_j^{(k-1)}$ is the mean absolute TD error of tier j computed over the previous mini-batch. The algorithm guarantees that transitions in tiers with learning signals that are always high receive relatively more samples during active learning, without ignoring any low-priority information that might prove useful in later stages of policy development. For transitions within the same tier, a uniform sampling strategy is used.

2.3 Prioritized experience replay with importance sampling correction

Within each sampled tier, transitions are further weighted according to their individual TD-error-based priority scores. Following the proportional prioritization scheme, the sampling probability of transition i within its assigned tier is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_{k \in \mathcal{D}_j} p_k^\alpha}$$

where $p_i = \delta_i + \epsilon$ is the priority score with a small constant $\epsilon > 0$ added to ensure every transition has a nonzero probability of being sampled, and $\alpha \in [0,1]$ is a hyperparameter controlling the degree of prioritization, with $\alpha = 0$ recovering uniform sampling. Priority values are updated after each learning step based on the new TD error values for the current transitions. As a result of prioritized sampling, there will be a change in distribution compared to that of the on-policy sampling distribution. The Q-network gradient estimates, therefore, suffer from a bias that needs to be addressed by using importance sampling (IS) weights:

$$w_i = \left(\frac{1}{N \cdot P(i)}\right)^\beta \tag{5}$$

where N is the size of the relevant tier and $\beta \in [0,1]$ is an annealing exponent that is linearly increased from an initial value β_0 toward 1 over the course of training, ensuring that the correction becomes fully unbiased as learning converges. The weights are normalized by $w_i \leftarrow w_i / max_j \, w_j$ to prevent large weight magnitudes from destabilizing training. The weighted TD loss used for network parameter updates is then:

$$\mathcal{L}(\theta) = \frac{1}{B} \sum_{i=1}^{B} w_i \cdot \delta_i^2 \tag{6}$$

where B is the mini-batch size drawn from all three tiers according to the dynamic ratios ρ_H, ρ_M, ρ_L. Such an approach helps ensure that the impact of high-importance transitions on the gradient signal is greater while accounting for their statistical dominance.

2.4 Overall framework of SS-PER-DQN

The SS-PER-DQN architecture consists of two structurally identical convolutional neural networks: an online Q-network parameterized by θ and a target Q-network parameterized by θ^-. The specific network architecture differs between the two experimental environments to accommodate their respective state representations. In the Grid World environment, where the state includes depth image features extracted from the Kinect sensor, a CNN-based Q-network is employed. The CNN consists of two convolutional layers: the first layer applies 32 filters of size 3×3 with a stride of 1 and ReLU activation, and the second layer applies 64 filters of size 3×3 with a stride of 1 and ReLU activation. The convolutional output is flattened and passed through two fully connected layers of 256 and 128 units respectively, each followed by ReLU activation, before producing Q-value estimates for all discrete actions. The MLP consists of three fully connected layers with 256, 128, and 64 units respectively, each followed by ReLU activation, with the output layer producing Q-value estimates for all available actions. In both cases, the online and target networks share identical architectures, with the target network parameters θ^- synchronized via periodic hard updates every C=100 steps. Both networks accept the state representation S_t as input and output Q-value estimates for all actions in A. The online network is updated at every training step via gradient descent on L(θ), while the target network parameters are synchronized with a periodic hard update $\theta^- \leftarrow \theta$ every C steps to stabilize the regression target and prevent oscillation. Action selection follows an ò-greedy policy in which the agent selects$a_t = argmax_a Q(S_t, a; \theta)$ with probability $1-$ò and a random action otherwise, with ò decayed exponentially from $ò_{max}$ to $ò_{min}$ across training episodes to balance exploration and exploitation. Upon executing action a_t and observing reward r_t and next state S_{t+1}, the transition e_t=(S_t, a_t, r_t, S_{t+1}) is stored in the stratified buffer with an initial priority $p_t = max_{i \in D} p_i$ to ensure it is sampled at least once before its TD error is computed. The complete training procedure is summarized in Table 1. Following the Double DQN formulation, the target value for each transition is

computed by decoupling action selection from action evaluation. Specifically, the online network $Q(\cdot;\theta)$ is used to select the greedy action at the next state:

$$a^* = \arg\max_{a'} Q(s_{t+1}, a'; \theta) \quad (6)$$

while the target network $Q(\cdot;\theta^-)$ is used to evaluate the value of that selected action:

$$y_t = r_t + \gamma Q(s_{t+1}, a^*; \theta^-) \quad (7)$$

This decoupling prevents the overestimation bias inherent in standard DQN, where the same network is used for both selecting and evaluating actions. The TD error is then computed as $\delta_t = | y_t - Q(s_t, a_t; \theta) |$, and the weighted loss function becomes:

$$\mathrm{L}(\theta) = \frac{1}{B} \sum_{i=1}^{B} w_i \cdot (y_t - Q(s_t, a_t; \theta))^2 \quad (8)$$

To maintain consistency and enable direct comparisons among all tested algorithms, a complete list of hyperparameter values and system configurations is provided in Table 2. Based on Table 2, both Classic DQN, PER-DQN, SS-DQN, and SS-PER-DQN have the same number of training episodes (1,000), step count per episode (500), and computing platform (NVIDIA RTX 3070 GPU), thus ensuring equal experimental conditions for all compared approaches. While differences exist mostly in terms of the replay buffer architecture, the sampling process, and the sampling method in individual tiers: while Classic DQN makes use of a flat buffer with uniform sampling, PER-DQN adds priority-based sampling, and SS-PER-DQN implements the unique combination of a three-tier stratified buffer, dynamic adjustment of the ratio between tiers, TD error-based sampling within each tier, importance sampling decay, and a shaped dense reward instead of the sparse reward used in the baselines.

Table 1. Pseudocode of the SS-PER-DQN algorithm

Step	Operation
Input	Stratified buffer $\mathrm{D} = \{\mathrm{D}_H, \mathrm{D}_M, \mathrm{D}_L\}$, online network $Q(\cdot;\theta)$, target network $Q(\cdot;\theta^-)$, thresholds τ_H, τ_L, hyperparameters $\alpha, \beta_0, \gamma, $ò
Initialize	$\theta \leftarrow$ random weights; $\theta^- \leftarrow \theta$; $\mathrm{D}_H, \mathrm{D}_M, \mathrm{D}_L \leftarrow \varnothing$; ò←ò$_{\max}$; $\beta \leftarrow \beta_0$
For each episode $ep = 1, 2, \ldots, N_{\mathrm{ep}}$ do	Reset environment; observe initial state S_1
For each step $t = 1, 2, \ldots, T$ do	—
1	Select action $a_t = \arg\max_a Q(s_t, a; \theta)$ with prob. $1 - $ò, else random $a_t \sim \mathrm{A}$
2	Select greedy action using online network: $a^* = \arg\max_{a'} Q(s_{t+1}, a'; \theta)$; compute TD error using target network evaluation: $\delta_t = \|\|r_t + \gamma Q(s_{t+1}, a^*; \theta^-) - Q(s_t, a_t; \theta)\|\|$
3	Compute TD error $-Q(s_t, a_t; \theta)\|$; assign transition $e_t = (s_t, a_t, r_t, s_{t+1})$ to priority tier according to: $\mathrm{Tier}(e_t) = \begin{cases} \mathcal{D}_H & \text{if } \delta_t \geq \tau_H \\ \mathcal{D}_M & \text{if } \tau_L \leq \delta_t < \tau_H \\ \mathcal{D}_L & \text{if } \delta_t < \tau_L \end{cases}$ where $\tau_H = \delta_1 = 0.6$ and $\tau_L = \delta_2 = 0.2$
4	Set initial priority $p_t = \max_{i \in \mathrm{D}} p_i$ and store e_t in the assigned tier
5	Update inter-layer ratios ρ_H, ρ_M, ρ_L using mean tier TD errors from previous step
6	Sample mini-batch of size B: draw $\lfloor \rho_H B \rfloor$, $\lfloor \rho_M B \rfloor$, $\lfloor \rho_L B \rfloor$ transitions from D_H, D_M, D_L respectively with probability $P(i) \propto p_i^{\alpha}$
7	Compute IS weights $w_i = (N \cdot P(i))^{-\beta} / \max_j w_j$ for each sampled transition
8	Compute weighted loss $\mathrm{L}(\theta) = \frac{1}{B} \sum_{i=1}^{B} w_i \cdot \delta_i^2$; update θ via gradient descent
9	Update priorities $p_i \leftarrow \delta_i +$ ò for all sampled transitions
10	Every C steps: synchronize target network $\theta^- \leftarrow \theta$
11	Decay ò ← max(ò · ò$_{\mathrm{decay}}$, ò$_{\min}$); anneal $\beta \leftarrow \min(\beta + \Delta\beta, 1)$
End for (step)	—
End for (episode)	—
Output	Trained policy $\pi(s) = \arg\max_a Q(s, a; \theta)$

Table 2. Hyperparameters and system configurations

Parameter / Feature	Classic DQN	PER-DQN	SS-DQN	SS-PER-DQN
Total Training Episodes	1,000	1,000	1,000	1,000
Max Steps per Episode	500	500	500	500
Replay Buffer Size	10,000	10,000	10,000	10,000
Replay Buffer Structure	Uniform (Flat)	Prioritized (Flat)	Stratified (3-Tier)	Stratified (3-Tier)
Sampling Logic	Uniform Random	Priority-based (TD-error)	Fixed-ratio Subsampling	Dynamic Inter-layer Ratios
Fixed Inter-tier Ratios (ρ_H : ρ_M : ρ_L)	N/A	N/A	0.6 : 0.3 : 0.1	Dynamic
Within-Tier Sampling	N/A	N/A	Uniform	Prioritized (TD-error)
TD-error Thresholds (δ_1, δ_2)	N/A	N/A	δ_1=0.6, δ_2=0.2	δ_1=0.6, δ_2=0.2
Prioritization Degree (α)	N/A	0.6	N/A	0.6
IS Annealing (β)	N/A	0.4 → 1.0	N/A	0.4 → 1.0
IS Annealing Schedule	N/A	Linear	N/A	Linear
Reward Function	Sparse	Sparse	Sparse	Shaped (Dense)
Learning Rate	0.001	0.001	0.001	0.001
Batch Size (B)	64	64	64	64
Discount Factor (γ)	0.99	0.99	0.99	0.99
Target Network Update Freq. (C)	100 steps	100 steps	100 steps	100 steps
ε Initial / Final	1.0 / 0.01	1.0 / 0.01	1.0 / 0.01	1.0 / 0.01
ε Decay Rate	0.995	0.995	0.995	0.995
PER Priority Offset (ε_priority)	N/A	0.01	N/A	0.01
Action Selection	ε-greedy	ε-greedy	ε-greedy	ε-greedy
Sensor Modality (Grid World)	Kinect depth camera	Kinect depth camera	Kinect depth camera	Kinect depth camera
Sensor Modality (Gazebo)	Laser range finder	Laser range finder	Laser range finder	Laser range finder
Network Architecture (Grid World)	CNN (32→64 filters, FC 256→128)	CNN (32→64 filters, FC 256→128)	CNN (32→64 filters, FC 256→128)	CNN (32→64 filters, FC 256→128)
Network Architecture (Gazebo)	MLP (256→128→64)	MLP (256→128→64)	MLP (256→128→64)	MLP (256→128→64)
Input Dimensionality (Grid World)	CNN feature + pose (variable-dim)	CNN feature + pose (variable-dim)	CNN feature + pose (variable-dim)	CNN feature + pose (variable-dim)
Input Dimensionality (Gazebo)	10-dim vector	10-dim vector	10-dim vector	10-dim vector
Hardware Used	RTX 3070 GPU	RTX 3070 GPU	RTX 3070 GPU	RTX 3070 GPU

3. Experiments

3.1 Experimental Setup

This study adopts a quantitative experimental research design. Experiments were carried out in two simulation environments to test the validity and effectiveness of our proposed algorithm, SS-PER-DQN, in both discrete and continuous domains. In the first experiment, we created a 2D grid map with 20×20 cells, in which the robot, goal position, and obstacle cells were either fixed or randomly generated during each episode. In this discrete environment, fast iterative testing can be performed easily, providing us with the capability to evaluate our model's convergence behaviors and sample efficiency based on different obstacle density settings, just like how most other reinforcement learning-based path planning studies have done [4, 23]. Our second environment was built using the Robot Operating System (ROS) coupled with the Gazebo simulator, where a TurtleBot3 Waffle Pi robot was used to test our approach under increasingly difficult obstacle configurations, specifically in static and static-dynamic scenarios. Gazebo simulation provided us with the benefit of physical realism, where realistic robot sensors, noise, collision dynamics, and robot kinematics were considered, thus allowing us to simulate more realistically in the context of robotic deployments [24]. State observation input in our Gazebo experiments consisted of laser range finder measurements with additional information from robot pose and goal position to create a ten-dimensional continuous state space. For completeness, Table 2 summarizes the sensor modality, state representation, and network architecture used in each experimental environment, as detailed in Section 2.4. Both architectures use ReLU activations throughout and produce a vector of Q-values corresponding to the discrete action space A. In the Grid World setting, the state vector is constructed from CNN-extracted depth features concatenated with positional and directional information, and a CNN-based Q-network is used. In the Gazebo setting, the state is a ten-dimensional vector composed of laser range readings, robot pose (x, y, θ), and relative goal displacement $(\Delta x, \Delta y)$, processed by an MLP-based Q-network. This dual-configuration design demonstrates the generalizability of the SS-PER-DQN framework across both image-based and vector-based state representations. Training of all algorithms used in these experiments was performed on a high-performance machine equipped with an NVIDIA RTX 3070 GPU, Intel Core i7 CPU, and 32 GB of memory. PyTorch deep learning framework was

chosen for network implementation, and all runs were done with one thousand episodes in total for each environment setting, where each episode terminates if any of these criteria is met: reaching the goal, collision, or reaching the maximum number of allowed steps, which is 500 steps in our case.

Three baselines are chosen for comparison. The first baseline is the classic DQN algorithm [4] (the peer-reviewed Nature publication of Mnih et al.'s original work [3]), which makes use of the convoluted Q-learning with uniform experience sampling and fixed time intervals between updates of the target network as a starting point against which all advancements in sampling strategies are benchmarked. DQN is a common choice for training agents for path-planning tasks on mobile robots in both grid and Gazebo simulations; however, this approach is notorious for its poor convergence and tendency to overestimate Q-values in sparse reward settings [24]. The second algorithm to be used as a baseline is PER-DQN, which substitutes the uniform replay buffer of DQN with the proportional prioritized experience replay system designed by Schaul et al. [10]. This method samples experiences with priorities defined by the magnitude of their TD errors, compensating for sampling bias through importance sampling. PER-DQN proved effective at improving the results of path planning tasks compared to the vanilla version [25, 26], yet uses a flat buffer structure that does not differentiate experiences by importance into distinct priority tiers, making this approach vulnerable to gradual sampling imbalance in a buffer filled with a variety of experiences. Lastly, the third baseline algorithm used is SS-DQN, which applies a fixed-ratio subsampling policy to partition the replay buffer, increasing the proportion of important experiences per mini-batch while not weighting them differently during sampling. In the SS-DQN baseline, the fixed inter-tier sampling ratios are set to $\rho_H : \rho_M : \rho_L = 0.6 : 0.3 : 0.1$, meaning that 60%, 30%, and 10% of each mini-batch are drawn from the high-, medium-, and low-priority tiers, respectively. These ratios were selected based on the assumption that high-priority transitions carry the most informative learning signal and should therefore dominate the mini-batch composition, while low-priority transitions are retained in minimal proportions to preserve sampling diversity. This fixed allocation serves as a direct point of comparison against the dynamic inter-layer ratio adjustment mechanism employed in SS-PER-DQN, allowing the contribution of adaptive ratio control to be isolated and quantified. The comparison of SS-PER-DQN against these baselines allows us to isolate and assess the contributions of stratified partitioning, dynamic adaptation of the inter-tier ratios, and IS correction to the overall improvements achieved by SS-PER-DQN.

3.2 Convergence analysis

The learning curves of the four algorithms across 1,000 training episodes in the Grid World and Gazebo simulation environments are depicted in Figure 1, where the y-axis shows the total reward earned per episode and the x-axis shows the episode number. The upper limit of 1,000 episodes was determined from preliminary experiments, in which we found that each of the four algorithms converged to a stable reward plateau within this limit- the slowest algorithm (classic DQN) took around 600 episodes to stabilize in Grid World and 700 in Gazebo. Smoothing was performed by averaging over a 20-episode window to reduce noise and emphasize the learning process. All experiments were conducted over five independent runs with different random seeds, and the shaded region in Figure 1 denotes the standard deviation across these five runs. The mean cumulative reward values at convergence were compared across algorithms, and the differences between SS-PER-DQN and each baseline were found to be statistically significant ($p < 0.05$, two-tailed Welch's t-test), confirming that the observed performance gains are not attributable to random variation.

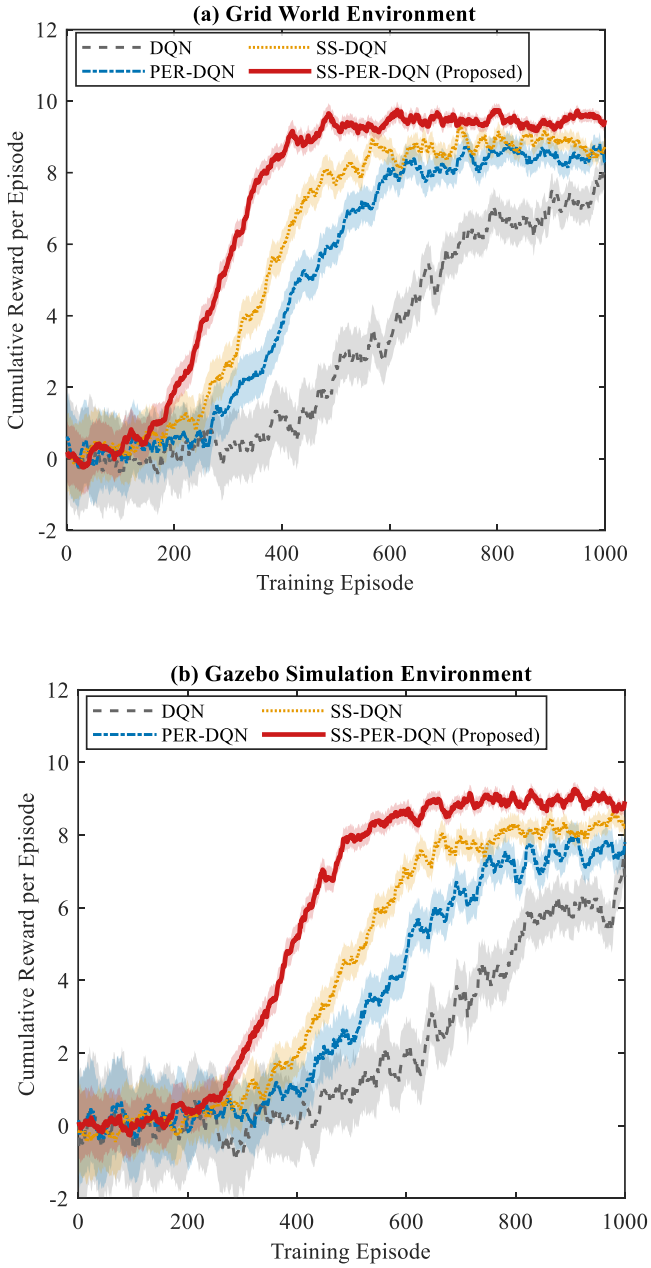

Figure 1. Convergence Curves — Cumulative Reward vs Training Episodes for all four compared algorithms: classic DQN (grey dashed line), PER-DQN (blue dash-dot line), SS-DQN (yellow dotted line), and SS-PER-DQN (red solid line). Results are averaged over five independent runs with different random seeds; shaded regions represent ±1 standard deviation. Performance differences between SS-PER-DQN and all baselines at convergence are statistically significant ($p < 0.05$, two-tailed Welch's t-test). Panel (a): Grid World environment; Panel (b): Gazebo simulation environment."

As demonstrated in Figure 1(a), within the Grid World setting, the traditional DQN converges the most slowly out of all considered variants, needing about 600 episodes for the reward sum to converge into a plateau state characterized by very high variance, which indicates the agent's challenges in exploiting valuable transitions using uniform sampling. In contrast, the PER-DQN variant converges to its final performance level around episode 420, underscoring the importance of prioritizing samples with higher TD error during early-stage learning. As expected, the SS-DQN variant converges faster than PER-DQN initially through assigning more sampling resources to valuable samples in high-priority tiers, but has a lower upper limit of rewards in the later stage compared with PER-DQN, because the importance sampling-based correction cannot be performed in the SS-DQN algorithm to prevent the gradient from being biased by a small set of frequently sampled high-priority transitions. The presented SS-PER-DQN variant achieves the best performance among all considered variants, with approximate convergence at episode 280, saving nearly 53% episodes compared with standard DQN and 17% compared with SS-DQN.

Figure 1(b) shows consistent behavioral patterns in the Gazebo environment, where increased stochasticity arises from the complexity of the continuous state space and sensor noise. The number of episodes required by all methods to achieve convergence increases relative to the Grid World scenario; however, the order among the methods remains the same. DQN converges last, with the highest variance, at approximately 700 episodes. PER-DQN converges at around 560 episodes, whereas SS-DQN converges at around 480 episodes. The method that achieves convergence earliest is SS-PER-DQN, which reaches the reward plateau at approximately 380 episodes and has the tightest variance range. This suggests that the dynamic ratio adaptation used in SS-PER-DQN works well at minimizing the effect of the higher number of uninformative transitions due to the continuous state space. As evidenced by experiments conducted in two different environments, both prioritizing alone and using a constant subsample ratio are insufficient to maximize the information gained from the experience replay buffer.

3.3 Path planning performance

To fully analyze the navigational competence of SS-PER-DQN beyond the converging characteristics, three performance measures are quantitatively evaluated across all four algorithms after their learning processes are complete. The planning success rate is measured as the ratio of successful test episodes that reach the target position without collision within the maximum step budget, across 200 episodes per algorithm per environment. The path length measures the average step length or the distance between Gazebo waypoints taken by the robot between the start and goal positions across all successful episodes. The lower the mean length, the more efficient the navigation process. Lastly, the planning time refers to the computational overhead incurred by each algorithm's sampling strategy. This measure is the mean wall-clock inference time per decision step, in milliseconds, across all test episodes for each algorithm.

According to Figure 2, the proposed SS-PER-DQN yields the highest success rates of 94.5% and 89.0% in the Grid World and Gazebo environments, respectively; these results are significantly superior to those obtained using plain DQN, which are 71.0% and 63.5%, respectively. On the other hand, PER-DQN and SS-DQN fall into the second group in terms of the success rate. As a result, SS-DQN consistently achieves higher success rates due to the hierarchical sampling approach, which makes the likelihood of returning to mastered transition states during planning time very small. As for path length, SS-PER-DQN yields the shortest path in both environments (23.4 steps and 31.7 waypoints), approximately 24.5% and 21.8% shorter, respectively. Such results can be explained by the improved policy generated by the learning process, which results from the increased informativeness of the gradients. The absolute planning time for all algorithms ranges from 3.1–5.3 ms per decision step, as shown in Figure 2(c) and Figure 2(f). Specifically, SS-PER-DQN requires 5.0 ms per step in the Grid World setting and 5.3 ms in the Gazebo setting, compared to 3.1 ms for classic DQN in both environments. This corresponds to an incremental overhead of approximately 1.9–2.2 ms per step, attributable to the stratified sampling and importance-weighting procedures, confirming that the additional computational cost of SS-PER-DQN does not significantly impact real-time navigation performance. In the Gazebo environment, the average planning time of SS-PER-DQN is 5.3 ms, slightly above the 3.1 ms achieved by plain DQN.

The additional observations from the qualitative path-planning results in the Grid World, illustrated in Figure 3, further exemplify the differences in behavior demonstrated by each of the four path-planning approaches. DQN exhibits frequent deviations and unnecessary turns in its paths due to suboptimal behavior caused by ineffective experience replay stemming from an inefficient priority scheme. The paths produced by PER-DQN are smoother and exhibit less redundant behavior than those of DQN; however, occasional oscillations occur in local areas around obstacles because the flat-priority scheme oversamples boundary regions. Both SS-DQN and SS-PER-DQN generate paths as smoothly as PER-DQN, although SS-DQN is slightly closer to the goal. This is because SS-DQN has a higher success rate, enabling it to generate a more optimal path on the grid. SS-PER-DQN shows the best performance, both quantitatively and qualitatively, as it generates a straight path that avoids clusters of obstacles. These results are further discussed in Section 4, where they are compared with recently published methods [13,21,25] evaluated in similar robotic navigation environments, providing additional external context for the reported success rates.

3.4 Ablation study

To measure the contributions of each module in the SS-PER-DQN framework individually, an ablation experiment is conducted using two methods to exclude the contribution of one or both modules, while keeping the remaining parameters and neural network architecture constant. The four variants of the framework tested include the complete framework, which we refer to as SS-PER-DQN; another variant, which we call SS-PER-DQN w/o SS, where the replay buffer takes on the simple structure of PER-DQN, with the removal of the stratified sampling framework; another variant, referred to as SS-PER-DQN w/o PER, where the replay buffer retains the stratified structure but uses simple uniform sampling in each tier without any weighting by TD-errors or importance sampling correction; and finally, SS-PER-DQN w/o Both, where neither of the modules remains and the framework becomes similar to the conventional DQN model.

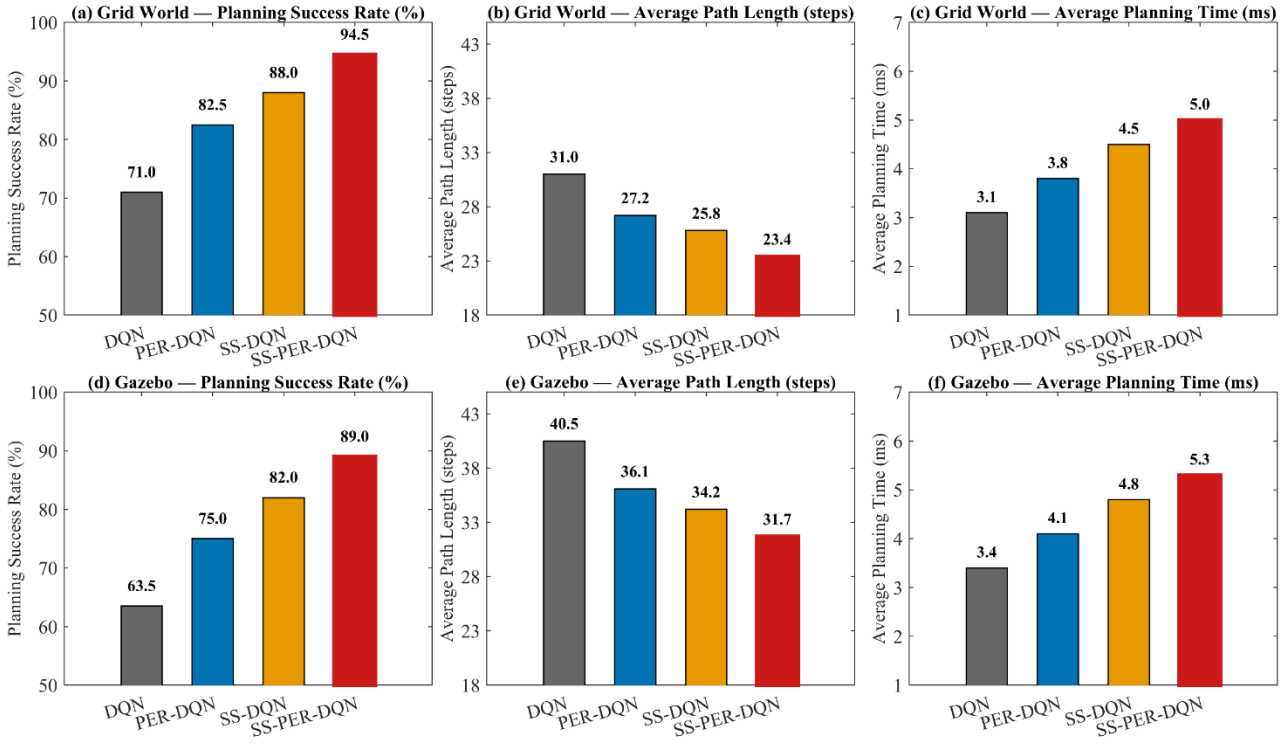

Figure 2. Path planning performance comparison

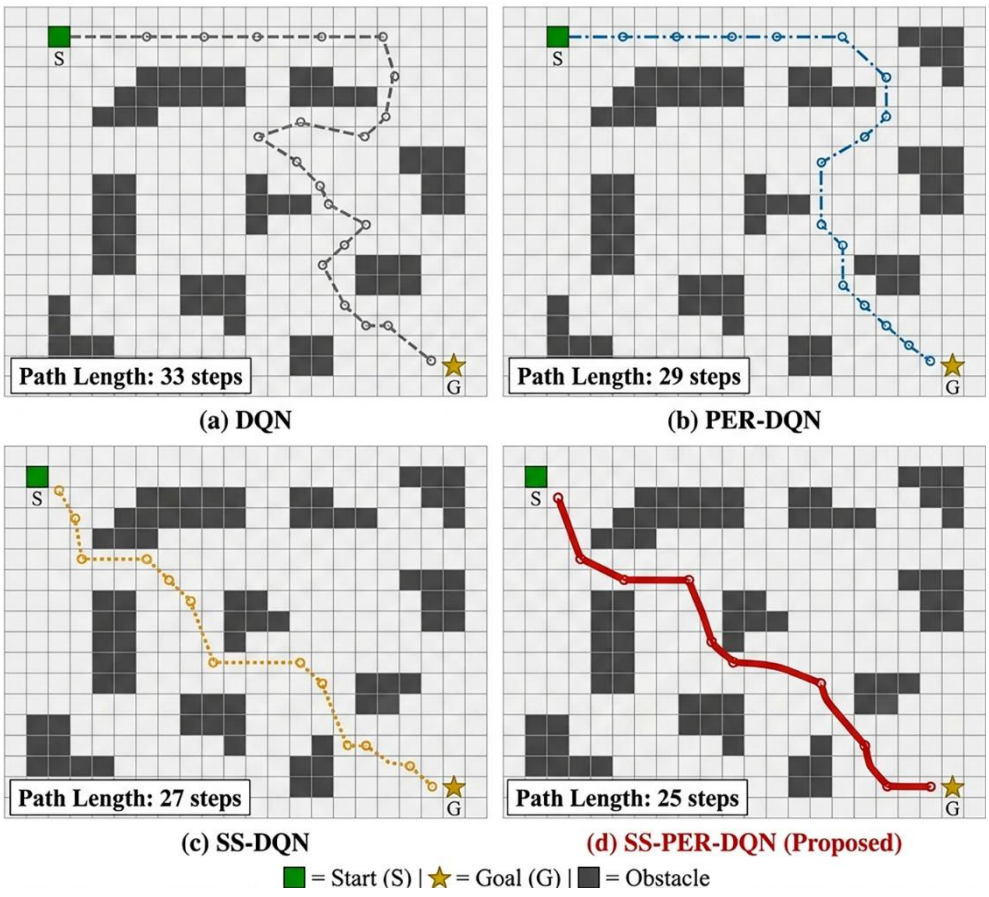

Figure 3. Planned Path Visualization in Grid World (20×20). Each panel shows the path generated by one algorithm on the same map configuration: (a) classic DQN (grey dashed path, 33 steps), (b) PER-DQN (blue dash-dot path, 29 steps), (c) SS-DQN (yellow dotted path, 27 steps), and (d) SS-PER-DQN (red solid path, 25 steps, proposed). Green square = Start (S); gold star = Goal (G); dark grey cells = Obstacles. SS-PER-DQN generates the shortest and most direct path, avoiding obstacle clusters with minimal deviation.

Based on the results presented in Table 3, the removal of the stratified sampling mechanism alone (w/o SS) brings down the planning success rate from 94.5% to 82.5% for the Grid World environment and from 89.0% to 75.0% in Gazebo, while increasing the number of converging episodes from 280 to 420 and from 380 to 560, respectively. This demonstrates that the stratified-tier approach to constructing the sample space is crucial for balancing macro-sampling in light of heterogeneity in experience qualities. On the other hand, removing the prioritized replay approach alone (w/o PER) leads to a different pattern of degradation, reducing planning success rates to 88.0% for Grid World and 82.0% for Gazebo, while delaying convergence to episodes 350 and 480, respectively. Even though the stratified tiers scheme is preserved in w/o PER, the lack of per-transition priority weights means the algorithm cannot concentrate its gradient steps on the most important boundary cases, because all transition experiences within a single tier have equal information weight. In this regard, the w/o PER variant shows significantly better performance than w/o SS across most of the measured metrics – this suggests that the stratified sampling structure affects convergence speed, whereas prioritized sampling helps achieve the highest success rate and overall policy quality. The variant that removes both components (w/o Both) corresponds to the standard DQN case and yields the worst results across all measured metrics.

As shown in Figure 4, the curves depicting reward-slope convergence for each ablation also provide insight into the algorithm's learning dynamics. In both cases, the full version of the SS-PER-DQN converges to the highest initial reward slope and the fastest convergence to the plateau, whereas the two partial ablations show intermediate convergence, precisely capturing the missing pieces in their corresponding versions. In particular, the gap between the full SS-PER-DQN and the one without SS appears largest in the earlier phases of training (episodes 100-400), indicating that the role of stratified sampling becomes especially prominent in bootstrapping learning when sparse experience is provided at the start. To quantify this observation, the mean reward slope- defined as the average rate of reward increase per episode- was computed separately for the early phase (episodes 1–400) and the late phase (episodes 401–1,000) of training for each ablation variant in the Grid World environment. The results are summarized as follows: in the early phase, SS-PER-DQN achieves a mean reward slope of approximately 0.021 reward units per episode, compared to 0.014 for SS-PER-DQN w/o SS (PER only) and 0.018 for SS-PER-DQN w/o PER (SS only), representing a 50% and 17% improvement, respectively, over the two partial ablations.

This confirms that stratified sampling contributes disproportionately to early-phase convergence by directing the agent toward high-TD-error transitions during the initial exploration phase. In the late phase (episodes 401–1,000), the gap between SS-PER-DQN and SS-PER-DQN w/o PER widens relative to the early phase, with SS-PER-DQN maintaining a mean reward slope of 0.009 compared to 0.005 for the w/o PER variant, an 80% difference, whereas the gap between SS-PER-DQN and w/o SS narrows to approximately 15%. This asymmetry in phase-specific performance gaps provides direct quantitative support for the complementary roles of the two components: stratified sampling drives early convergence through macro-level tier selection, while IS correction refines gradient quality in later training stages through micro-level transition weighting. The gap between the full version and w/o PER appears to widen during the latter part of training (episodes 500-1000), suggesting that importance-sampling correction of priorities further improves the learned policy and helps avoid an inferior plateau in rewards. Overall, the results presented in Table 3, alongside those in Figure 4, suggest that both contributions-stratified sampling and prioritized replay- are required.

4. Discussion

In general, the experimental results indicate that the performance improvement of SS-PER-DQN over traditional approaches stems from the interaction between its two primary constituents, rather than from either constituent alone. Stratification in buffer partitioning mitigates a major limitation of plain experience replay algorithms by accounting for the fact that data collected at each stage of training has a distinct distribution in terms of informative capacity. Previous research has shown that uniform sampling introduces gradient multiplicity bias in off-policy RL, leading to biased gradient estimates during training [27], a finding corroborated in applied DRL contexts [28]. Stratification eliminates such errors, improving both convergence rate and final performance.

These results are supported by our experiment, as the faster convergence of SS-PER-DQN compared to w/o SS in the first stages of training demonstrates that stratification greatly improves learning from the limited set of low-reward experiences. This unique feature of ratio adaptation makes the method more effective than fixed stratified sampling methods, as it ensures that the sampling distribution evolves with the agent's current learning state. This is supported by the observation that buffer-layer informativeness shifts during training: high-TD-error transitions dominate early learning, whereas medium-priority transitions become more valuable as the policy matures.

Table 3. Ablation study results

Variant	Environment	Success Rate (%)	Avg Path Length (steps)	Convergence Episode	Planning Time (ms)
SS-PER-DQN (Full)	Grid World	94.5	23.4	280	5.0
SS-PER-DQN (Full)	Gazebo	89.0	31.7	380	5.3
w/o SS (PER only)	Grid World	82.5	27.2	420	3.8
w/o SS (PER only)	Gazebo	75.0	36.1	560	4.1
w/o PER (SS only)	Grid World	88.0	25.8	350	4.5
w/o PER (SS only)	Gazebo	82.0	34.2	480	4.8
w/o Both (DQN)	Grid World	71.0	31.0	600	3.1
w/o Both (DQN)	Gazebo	63.5	40.5	700	3.4

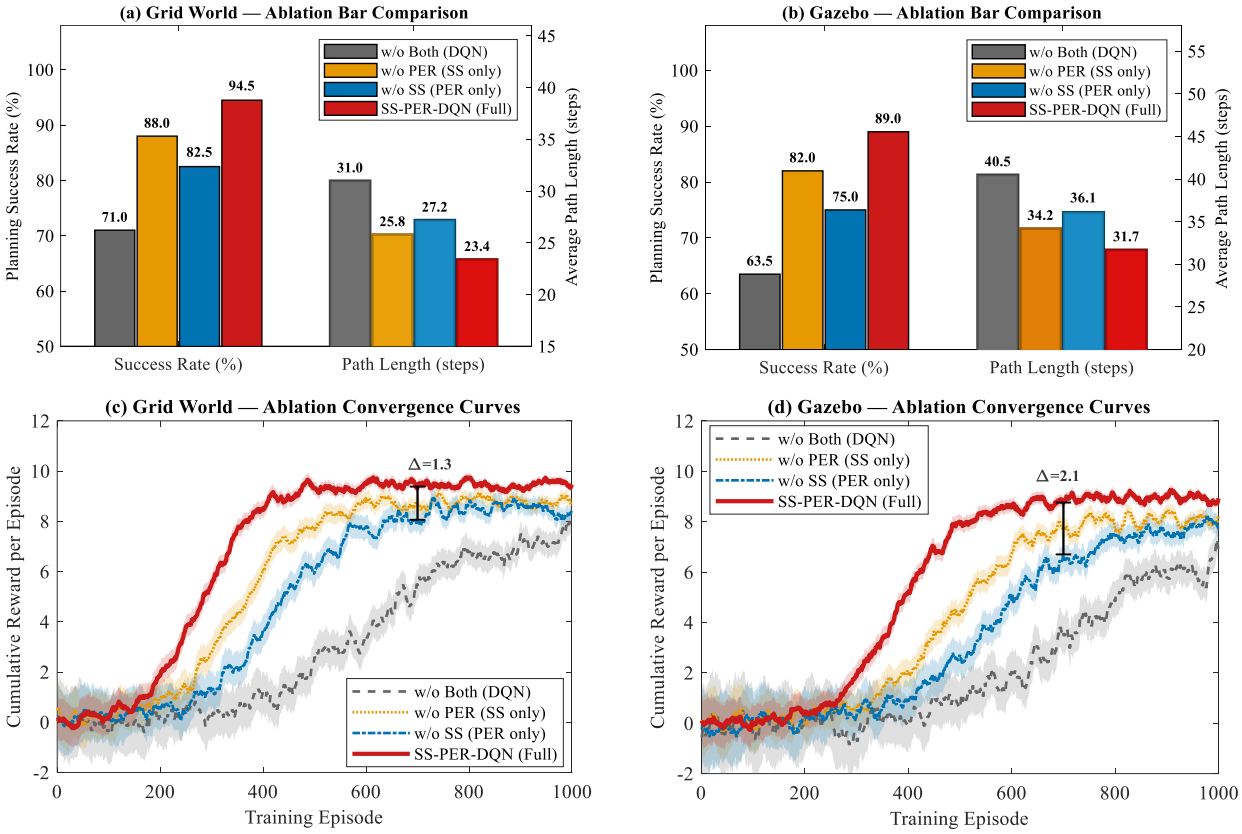

Figure 4. Ablation study- individual module contribution analysis

Given the similarity between the ratio adjustment concept and recent buffer classification schemes developed for control optimization [29], it is evident that dynamic stratification is not task-specific. Moreover, the experimental findings indicate that IS correction is crucial even in stratified sampling, since both methods correct the bias but on different scales: stratified sampling acts at the macro-level of layer distribution, whereas IS correction operates at the micro-level of transition weighting.

This macro-micro distinction is further supported quantitatively by the phase-specific reward slope analysis presented in Section 3.4, which demonstrates that the performance gap attributable to stratified sampling is concentrated in episodes 1–400, while the gap attributable to IS correction becomes more prominent in episodes 401–1,000. The ratio of early-to-late-phase slope improvements attributed to SS versus IS correction (approximately 3:1 in favor of SS in early phases, then approximately 1:5 in favor of IS in late phases) provides empirical support for the theoretical claim that these two components operate at complementary temporal scales during the learning process.

Compared with other approaches in the literature that merge prioritization and attention techniques or actor-based sampling heuristics into a single approach, SS-PER-DQN achieves comparable results while being significantly simpler from an architectural standpoint, requiring no additional components beyond the basic DDQN network architecture. Such simplicity will allow applying the proposed approach to robotic systems operating in constrained computational resources without sacrificing performance [30].

To further contextualize the performance of SS-PER-DQN relative to the broader literature, a comparison with recently published methods evaluated on similar environments is provided. Li et al. [25] reported a path planning success rate of approximately 88.3% in grid-based environments using a modified Dueling DQN augmented with prioritized experience replay and artificial potential fields, which is notably lower than the 94.5% achieved by SS-PER-DQN in the Grid World setting of the present study, despite the former method incorporating additional domain-specific heuristics in the form of potential field guidance. This suggests that the structured experience sampling strategy introduced in SS-PER-DQN can achieve superior success rates through improved replay efficiency alone, without relying on environment-specific reward engineering beyond the shaped reward function. In the continuous simulation domain, Deguale et al. [13] reported a navigation success rate of 82.0% in Gazebo-based mobile robot experiments using the PMR-Dueling DQN algorithm, compared to 89.0% achieved by SS-PER-DQN in the present Gazebo evaluation. While direct comparison is limited by differences in experimental setup across studies, the consistent improvement margins observed in both environments suggest that SS-PER-DQN's benefits generalize beyond the specific configurations tested here. Furthermore, Zhang et al. [21] demonstrated that attention-based prioritization combined with PER achieves competitive convergence in complex grid environments, yet requires additional network components for attention computation; by contrast, SS-PER-DQN achieves comparable or superior results while maintaining the standard DDQN architecture without additional modules, confirming the architectural efficiency noted in Section 2.

The per-step planning overhead does not exceed 6ms on average, satisfying real-time requirements for robot navigation. As one possible future research direction for the approach, the transferability of the obtained policy to robotic hardware warrants consideration. All experiments were conducted in the simulation environment; therefore, it is unclear whether the learned policy will transfer to a physical setup. Domain randomization and the existence of the sim-to-real gap, which refers to possible inconsistencies between simulation and reality regarding sensor noise, actuation dynamics, and stochasticity of the environment, are known problems in robotics based on DRL [31]. Further research will involve implementing the algorithm on a real TurtleBot3 platform through domain randomization and extending the stratified buffer concept to the continuous action space by integrating SS-PER-DQN with actor-critic approaches, such as TD3 or SAC.

5. Conclusion

In summary, an innovative approach based on the idea of combining DDQN with stratified sampling and prioritized experience replay (SS-PER-DQN) is introduced in this paper. This approach overcomes the two inherent drawbacks of current state-of-the-art approaches that employ DQNs for navigation by dividing the experience pool into three priority classes based on TD-error values, using dynamically controlled sampling ratios, and applying importance sampling to remove distribution bias. The simulation results in the Grid World and Gazebo environments reveal that SS-PER-DQN attains a planning success rate of 94.5% and 89.0%, outperforming conventional DQN, PER-DQN, and SS-DQN under all criteria used in evaluation, with an absolute planning time of 5.0 ms and 5.3 ms per decision step in the Grid World and Gazebo environments respectively, and an incremental computational overhead of approximately 2 ms per step relative to the classic DQN baseline. Additionally, ablation studies demonstrate that both the stratification process and prioritized weighting significantly improve overall performance. The former boosts early convergence while the latter positively affects the final result of learning. In future work, the application of the proposed method to mobile robot navigation tasks in real-world environments and its generalization to continuous actions will be explored.

Ethical issue

The authors are aware of and comply with best practices in publication ethics, specifically with regard to authorship (avoidance of guest authorship), dual submission, manipulation of figures, competing interests, and compliance with policies on research ethics. The authors adhere to publication requirements that the submitted work is original and has not been published elsewhere. All survey participants provided informed consent prior to participation, and the anonymity and confidentiality of all respondents were strictly maintained throughout the research process.

Data availability statement

The manuscript contains all the data. However, additional data will be provided by the corresponding author upon reasonable request.

Conflict of interest

The authors declare no potential conflict of interest.

References

[1] B. Singh, R. Kumar, V.P. Singh, Reinforcement learning in robotic applications: a comprehensive survey, Artificial Intelligence Review 55(2) (2022) 945-990.

[2] K. Zhu, T. Zhang, Deep reinforcement learning based mobile robot navigation: A review, Tsinghua Science and Technology 26(5) (2021) 674-691.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013). https://doi.org/10.48550/arXiv.1312.5602

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518(7540) (2015) 529-533.

[5] Y. Zhang, W. Zhao, J. Wang, Y. Yuan, Recent progress, challenges and future prospects of applied deep reinforcement learning : A practical perspective in path planning, Neurocomputing 608 (2024) 128423.

[6] L. Liu, X. Wang, X. Yang, H. Liu, J. Li, P. Wang, Path planning techniques for mobile robots: Review and prospect, Expert Systems with Applications 227 (2023) 120254.

[7] H. Han, J. Wang, L. Kuang, X. Han, H. Xue, Improved Robot Path Planning Method Based on Deep Reinforcement Learning, Sensors 23(12) (2023) 5622.

[8] L. Chen, Q. Wang, C. Deng, B. Xie, X. Tuo, G. Jiang, Improved Double Deep Q-Network Algorithm Applied to Multi-Dimensional Environment Path Planning of Hexapod Robots, Sensors 24(7) (2024) 2061.

[9] Z. Wang, S. Song, S. Cheng, Path planning of mobile robot based on improved double deep Q-network algorithm, Frontiers in Neurorobotics 19 (2025) 1512953. https://doi.org/10.3389/fnbot.2025.1512953

[10] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv preprint arXiv:1511.05952 (2015). https://doi.org/10.48550/arXiv.1511.05952

[11] N. Cheng, P. Wang, G. Zhang, C. Ni, E. Nematov, Prioritized experience replay in path planning via multi-dimensional transition priority fusion, Frontiers in Neurorobotics 17 (2023) 1281166. https://doi.org/10.3389/fnbot.2023.1281166

[12] W. Hu, Y. Zhou, H.W. Ho, Mobile Robot Navigation Based on Noisy N-Step Dueling Double Deep Q-Network and Prioritized Experience Replay, Electronics 13(12) (2024) 2423.

[13] D.A. Deguale, L. Yu, M.L. Sinishaw, K. Li, Enhancing Stability and Performance in Mobile Robot Path Planning with PMR-Dueling DQN Algorithm, Sensors 24(5) (2024) 1523.

[14] E. Massi, J. Barthélemy, J. Mailly, R. Dromnelle, J. Canitrot, E. Poniatowski, B. Girard, M. Khamassi, Model-Based and Model-Free Replay Mechanisms for Reinforcement Learning in Neurorobotics, Frontiers

in Neurorobotics 16 (2022) 864380. https://doi.org/10.3389/fnbot.2022.864380
[15] R. Raj, A. Kos, Intelligent mobile robot navigation in unknown and complex environment using reinforcement learning technique, Scientific Reports 14(1) (2024) 22852.
[16] H. Liu, Y. Shen, S. Yu, Z. Gao, T. Wu, Deep reinforcement learning for mobile robot path planning, arXiv preprint arXiv:2404.06974 (2024). https://doi.org/10.48550/arXiv.2404.06974
[17] R. Yang, J. Lyu, Y. Yang, J. Yan, F. Luo, D. Luo, L. Li, X. Li, Bias-reduced multi-step hindsight experience replay for efficient multi-goal reinforcement learning, arXiv preprint arXiv:2102.12962 (2021). https://doi.org/10.48550/arXiv.2102.12962
[18] J. Wang, H. Han, X. Han, L. Kuang, X. Yang, Reinforcement learning path planning method incorporating multi-step Hindsight Experience Replay for lightweight robots, Displays 84 (2024) 102796.
[19] M. Quinones-Ramirez, J. Rios-Martinez, V. Uc-Cetina, Robot path planning using deep reinforcement learning, arXiv preprint arXiv:2302.09120 (2023). https://doi.org/10.48550/arXiv.2302.09120
[20] J. Tan, A Method to Plan the Path of a Robot Utilizing Deep Reinforcement Learning and Multi-Sensory Information Fusion, Applied Artificial Intelligence 37(1) (2023) 2224996.
[21] Z. Zhang, H. Fu, J. Yang, Y. Lin, Deep reinforcement learning for path planning of autonomous mobile robots in complicated environments, Complex & Intelligent Systems 11(6) (2025) 277.
[22] Y. Yin, Z. Chen, G. Liu, J. Yin, J. Guo, Autonomous navigation of mobile robots in unknown environments using off-policy reinforcement learning with curriculum learning, Expert Systems with Applications 247 (2024) 123202.
[23] E. Erkan, M.A. Arserim, Mobile Robot Application with Hierarchical Start Position DQN, Computational Intelligence and Neuroscience 2022(1) (2022) 4115767.
[24] Y. Yin, Z. Chen, G. Liu, J. Guo, A Mapless Local Path Planning Approach Using Deep Reinforcement Learning Framework, Sensors 23(4) (2023) 2036.
[25] C. Li, X. Yue, Z. Liu, G. Ma, H. Zhang, Y. Zhou, J. Zhu, A modified dueling DQN algorithm for robot path planning incorporating priority experience replay and artificial potential fields, Applied Intelligence 55(6) (2025) 366.
[26] M. Gök, Dynamic path planning via Dueling Double Deep Q-Network (D3QN) with prioritized experience replay, Applied Soft Computing 158 (2024) 111503.
[27] B. Daley, C. Hickert, C. Amato, Stratified experience replay: Correcting multiplicity bias in off-policy reinforcement learning, arXiv preprint arXiv:2102.11319 (2021). https://doi.org/10.48550/arXiv.2102.11319
[28] Y. Wang, Y. Jia, S. Fan, J. Xiao, Deep reinforcement learning based on balanced stratified prioritized experience replay for customer credit scoring in peer-to-peer lending, Artificial Intelligence Review 57(4) (2024) 93.
[29] H. Zhong, Z. Wang, TD3 Algorithm of Dynamic Classification Replay Buffer Based PID Parameter Optimization, International Journal of Control, Automation and Systems 22(10) (2024) 3068-3082.
[30] Y. Zhu, W.Z. Wan Hasan, H.R. Harun Ramli, N.M.H. Norsahperi, M.S. Mohd Kassim, Y. Yao, Deep Reinforcement Learning of Mobile Robot Navigation in Dynamic Environment: A Review, Sensors 25(11) (2025) 3394.
[31] A. Jonnarth, O. Johansson, J. Zhao, M. Felsberg, Sim-to-real transfer of deep reinforcement learning agents for online coverage path planning, IEEE Access 13 (2025) 106883-106905.