N-DQN-RRT: A Hybrid Deep Q-Network Guided RRT Algorithm for an Efficient Path Planning in Two-Dimensional Environments

Wai Yan Naing Win[1], K. Kanagalakshmi[2]*, K. Priyadharshini[3],
Zeyad A. T Ahmed[4], R. RajiniGanth[5], P. Ganesh Kumar[6], Atif Mahmood[7]*
Faculty of Data Science and Information Technology,
INTI International University, Nilai, 71800, Negeri Sembilan, Malaysia[1,4]
School of Science and Computer Studies, CMR University, Bangalore, Karnataka, India[2]
Jeppiaar Engineering College, Affiliated with Anna University, Chennai, Tamil Nadu, India[3]
SNS College of Engineering, Department of Computer Science and Technology, Coimbatore, India[5]
Department of Computer Science and Engineering, College of Engineering, Anna University, Chennai, Tamil Nadu, India[6]
Department of Computer System and Technology-Faculty of Computer Science and Information Technology,
Universiti Malaya, Kuala Lumpur, Malaysia[7]

***Abstract*—Autonomous navigation requires path-planning algorithms that can find safe and efficient routes while avoiding obstacles. Classical sampling-based planners such as Rapidly-exploring Random Tree (RRT) and RRT* are widely used because they are flexible and easy to apply in different environments. However, these methods often produce longer paths and require many samples in complex spaces. This paper proposes Neural Deep-Q Network RRT (N-DQN-RRT), a hybrid path-planning algorithm that integrates a Deep Q-Network (DQN) agent into an RRT*-style planning framework. The proposed method uses a directed-random sampling strategy. In this strategy, 75% of samples are guided by the trained DQN policy, 5% are assigned to goal bias, and 20% are sampled uniformly at random. The random component is retained to reduce premature policy bias and maintain exploration. The planner also uses an adaptive rewiring radius that changes according to local obstacle conditions. This allows the method to improve local path quality while avoiding unnecessary neighbor searches in cluttered regions. The algorithm was evaluated in three static two-dimensional environments with increasing map sizes of 10×10, 25×25, and 50×50. Each environment used randomized obstacle layouts and 100 independent trials, giving 300 trials per algorithm. The proposed method was compared with RRT and RRT* using six metrics: path length, steering angle, iteration count, sampling size, computation time, and success rate. Results were reported as mean $\pm$ standard deviation with 95% confidence intervals. N-DQN-RRT achieved a 100% success rate across all 300 trials, while RRT achieved 10–100% and RRT* achieved 14–100%, depending on map complexity. In the largest environment, N-DQN-RRT reduced path length by 11.9%, steering angle by 49.5%, sampling size by 75.1%, and computation time by 91.9% compared with RRT*. These results show that DQN-guided sampling can improve the empirical efficiency of RRT*-style planning in static two-dimensional obstacle environments.**

***Keywords*—Path planning; Rapidly-Exploring Random Trees (RRT); RRT*; Deep Q-Network (DQN); deep reinforcement learning; resilient infrastructure**

I. Introduction

Deep learning has been successfully applied across a wide range of intelligent perception and decision-making tasks, demonstrating its capability to learn complex representations from data [1]. Autonomous navigation is an important part of modern mobile robotics. A mobile robot must move from a start position to a target position while avoiding obstacles. This task is known as path planning. It is still a difficult problem because the planner must find a safe path under limited time and computational resources [2], [3].

The planning environment strongly affects the choice of method. In a static environment, obstacles remain fixed during planning. The robot can plan using a known map of the configuration space. In a dynamic environment, obstacles may move or appear during operation. This makes planning more difficult because the robot may need to update or replan its route during execution [3], [2]. For this reason, efficient and reliable path planning remains a key requirement for autonomous robots.

Classical graph-search methods such as A*, Dijkstra, and breadth-first search use a discretized representation of the environment. These methods are effective when the search space is small and the map resolution is manageable. However, their memory use and search cost increase as the map becomes larger or more detailed. This makes them less practical for large planning spaces and high-resolution maps [4], [2].

Sampling-based planners address this limitation in a different way. They do not need to search the full configuration space explicitly. Instead, they build a roadmap or tree by sampling points from the free space and connecting them through local collision checks [5], [2]. Rapidly-exploring Random Tree, or RRT, is one of the most common sampling-based planners. It grows a tree from the start position by repeatedly sampling the environment and extending the tree toward the sampled point. RRT is useful because it can find feasible paths in complex spaces.

*Corresponding author.

However, its paths are often irregular and longer than necessary. Its performance also depends strongly on random sampling [4], [2].

Several RRT variants have been proposed to improve path quality and convergence. RRT* adds parent selection and rewiring to improve path cost over time. Bi-directional RRT* grows two trees from the start and goal regions. Informed RRT* limits sampling after an initial solution is found so that search is focused on a more useful region of the space [6], [7]. These methods improve classical RRT in many cases. Still, they rely heavily on hand-designed sampling rules. They do not learn from previous planning experience.

Learning-based planning has been studied as a way to guide sampling toward more promising regions. Neural and generative models can learn useful structure from maps or previous paths. These learned priors can reduce unnecessary exploration and improve planning efficiency [8], [9]. Deep reinforcement learning also offers a useful framework for navigation. An agent can learn a policy through rewards and penalties. This allows the planner to prefer actions that move toward the goal and avoid obstacles [10], [11], [12].

Deep Q-Network, or DQN, is a value-based reinforcement learning method. It is suitable when the action space can be represented by a finite set of actions. In path planning, this makes DQN useful for selecting movement directions or guiding sample generation. However, a learned policy alone may be biased by its training distribution. It may also fail when the test environment differs from the training maps. For this reason, a hybrid planner can be useful. It can combine learned guidance with a random sampling component.

This paper proposes N-DQN-RRT, a hybrid path-planning algorithm for static two-dimensional environments. The method integrates a trained DQN agent into an RRT*-style planning framework. The DQN guides most sampling decisions toward promising directions. A random sampling component is retained to preserve exploration behavior and reduce policy bias. The planner also uses an adaptive rewiring radius. The radius is reduced near cluttered regions and expanded in more open regions. This allows the planner to improve path quality while reducing unnecessary neighbor searches.

The main contribution of this work is an empirical evaluation of the proposed N-DQN-RRT planner. The algorithm is tested in randomized two-dimensional obstacle environments with increasing size and complexity. Its performance is compared with classical RRT and RRT*. The evaluation uses path length, steering angle, iteration count, sampling size, computation time, and success rate. The goal is not to prove formal optimality or probabilistic completeness. Instead, the goal is to show whether DQN-guided sampling and adaptive rewiring can improve practical planning performance under the tested simulation conditions.

The contributions of this paper are summarized as follows:

- A hybrid DQN-guided RRT* planner is proposed for static two-dimensional path planning.
- The planner uses a seven-feature state representation that includes goal direction, obstacle proximity, and local free-space information.
- An 80/20 guided-random sampling strategy is used to balance learned guidance and stochastic exploration.
- An adaptive rewiring radius is introduced to adjust local optimization according to obstacle conditions.
- The method is evaluated across three randomized map sizes and compared with RRT and RRT* using six performance metrics.

II. Related Work

Path planning has been studied through classical search methods, sampling-based planners, learning-guided planners, and reinforcement learning approaches. This section reviews these groups and shows the gap addressed by the proposed N-DQN-RRT method (Table I).

A. Classical Sampling-Based Path Planning

Sampling-based planners are widely used in mobile robot path planning. They are useful when the configuration space is large or difficult to describe fully. Instead of building a complete grid of the space, these methods sample points from the free region and connect them through local collision checks [13], [2].

Rapidly-exploring Random Tree, or RRT, is one of the most common sampling-based planners. It grows a tree from the start point toward randomly sampled points. This makes RRT suitable for single-query planning problems. It can often find a feasible route in complex spaces. However, the path is usually not smooth or short. The result also depends strongly on the random samples [14], [4].

RRT* improves RRT by adding parent selection and rewiring. These steps reduce the path cost as the tree grows. Still, RRT* may require many samples before it converges to a high-quality path. Other variants improve the search process in different ways. Bi-directional RRT* grows two trees from the start and goal regions. Informed RRT* limits the sampling region after a first solution is found. Hybrid RRT* methods also combine global sampling with local improvement rules [6], [7]. These methods improve classical RRT, but they still depend on fixed sampling rules. They do not learn from earlier planning experience.

B. Learning-Guided Sampling-Based Planning

Learning-based planning tries to reduce unnecessary exploration. The main idea is simple. A model learns where useful samples are likely to appear. The planner can then spend more effort in promising regions.

Generative models have been used for this purpose. For example, GAN-based planners can use an environment image to predict a region where a feasible path is likely to exist. This region is then used as a sampling guide for a classical planner [8]. Such methods can improve convergence in tested environments. However, their performance depends on the learned map representation and the similarity between training and test environments.

Neural sampling methods have also been used with RRT-style planners. Neural RRT* uses a convolutional neural

TABLE I. SUMMARY OF RELATED WORK ON SAMPLING-BASED, LEARNING-BASED, AND DRL-BASED PATH PLANNING

Research group	Representative methods	Main idea	Key limitations	Related references
Classical sampling-based planning	RRT, RRT*, PRM	Builds a tree or roadmap through random samples in the free space. These methods are useful when the full configuration space is difficult to search directly.	Path quality can be poor. Convergence may be slow. Performance depends strongly on random samples.	[14], [4], [2]
Improved RRT variants	Bi-RRT*, Informed RRT*, hybrid RRT*	Improves classical RRT by using rewiring, bidirectional search, or restricted sampling after an initial path is found. These methods can improve path cost and search efficiency.	Still depends on hand-designed sampling rules. It does not learn from previous planning experience. Performance may degrade in cluttered or narrow regions.	[6], [7]
Learning-guided sampling	GAN-RRT, Neural RRT*, DNN-guided sampling	Uses neural or generative models to guide samples toward promising regions of the map. This can reduce unnecessary exploration and improve planning efficiency.	Requires training data. The learned sampler may be sensitive to changes between training and test environments. Generalization remains a concern.	[8], [9], [15]
Deep reinforcement learning planning	DQN, PPO, SAC, TD3	Learns navigation behavior from interaction with the environment. The reward function encourages goal progress and penalizes unsafe actions.	Training can be expensive. Results depend on reward design and hyperparameters. Some methods have difficulty transferring from simulation to real robots.	[10], [16], [17], [18], [19], [20]
Hybrid RL and RRT planning	RL-RRT, DRL-RRT, RRT*-RL	Combines learned guidance with the tree structure of RRT or RRT*. The planner keeps classical expansion behavior while using learning to reduce inefficient exploration.	May lose formal planning guarantees if learning dominates sampling. A learned policy can be biased by the training distribution. Ablation is needed to show which component gives the improvement.	[11], [12], [21]
Proposed direction	N-DQN-RRT	Uses a DQN-guided sampler inside an RRT*-style framework. It keeps random sampling for exploration and adds an adaptive rewiring radius for local path improvement.	Current evaluation is limited to static two-dimensional environments. Dynamic scenes, cross-map transfer, and hardware validation remain future work.	This study

network to guide non-uniform sampling for optimal path planning [9]. Deep neural network enhanced sampling has also been studied in three-dimensional planning spaces [15]. These studies show that learned priors can improve sample efficiency. At the same time, they also show a common limitation. A learned sampler may lose reliability when the test environment differs from the training data.

C. Deep Reinforcement Learning for Robot Navigation

Deep reinforcement learning has become a common tool for robot navigation. In this setting, an agent learns from interaction with the environment. The reward function encourages goal progress and penalizes unsafe behavior. This makes DRL useful for navigation tasks where direct planning is difficult [10], [16].

Value-based methods such as Deep Q-Network are useful when the action space is discrete. A DQN can learn the value of each possible action from the current state. This is suitable for planners that choose among a fixed set of movement directions. Improved DQN variants have been studied for mobile robot path planning, often by changing the reward function or network structure [17], [22]. However, DQN methods can be sensitive to reward design. They may also suffer when the learned policy is applied outside the training distribution.

Policy-gradient and actor-critic methods are also common in navigation. Methods such as PPO, SAC, and TD3 have been applied to dynamic path planning and obstacle avoidance [18], [19], [20]. These methods can handle more complex action spaces. They can also learn flexible behavior in dynamic scenes. Their main drawbacks are training cost, hyperparameter sensitivity, and the gap between simulation and real deployment.

D. Hybrid Planning with Reinforcement Learning and RRT

A different line of work combines learning with classical sampling-based planners. This is useful because the planner keeps the tree structure of RRT while the learning model guides part of the search. RL-RRT is one example. It uses a learned policy and a reachability estimator to guide tree growth in kinodynamic planning [11]. Other DRL-integrated RRT methods use learned policies to improve sample selection or path expansion [12]. Recent work has also explored reinforcement learning inside RRT*-style frameworks for mobile robot planning [21].

These hybrid methods are important because they do not fully replace classical planning. Instead, they add learning where random exploration is inefficient. This helps preserve the structure of sampling-based planning while adding experience-based guidance. However, many existing studies use different environments, different learning models, and different evaluation settings. This makes direct comparison difficult.

E. Research Gap

The reviewed studies show that classical RRT methods are flexible but sample inefficient. They also show that learned sampling can improve planning speed and path quality. However, there is still a need for a simple value-based hybrid planner that keeps the RRT* structure and uses DQN guidance in a controlled way.

TABLE II. PLANNER PARAMETERS FOR THE THREE MAP SETTINGS

Parameter	Map 1	Map 2	Map 3
Environment size	10×10	25×25	50×50
Step size η	1.0	1.0	1.0
Maximum iterations	250	500	1500
Goal radius	1.0	1.0	1.0
Start position	$(1, 1)$	$(1, 1)$	$(1, 1)$
Goal position	$(9, 9)$	$(24, 24)$	$(49, 49)$
Base rewire radius	2.0	2.0	2.0

This paper addresses that gap through N-DQN-RRT. The method uses a DQN-guided sampler inside an RRT*-style framework. It also keeps a random sampling component to reduce policy bias. An adaptive rewiring radius is added to improve local path quality. The method is evaluated in static two-dimensional random obstacle environments with increasing map size. The goal is to test whether learned guidance can improve practical planning performance when compared with classical RRT and RRT*.

III. METHODOLOGY

This section describes the experimental setup and the proposed N-DQN-RRT algorithm. The study uses a simulation-based design. The proposed method is compared with classical RRT and RRT*. All planners are evaluated under the same map settings. They also use the same start and goal positions.

A. Experimental Framework

The experiments use a quantitative simulation framework. Three algorithms are evaluated. These are RRT, RRT*, and the proposed N-DQN-RRT. The goal is to compare their planning performance in static two-dimensional environments.

All algorithms are implemented in Python. NumPy is used for numerical operations. Matplotlib is used for visualization. PyTorch is used to build and train the DQN model. SciPy is used for statistical analysis. The experiments are run on a Google Colab CPU environment.

Each map configuration is tested using 100 independent trials. No fixed seed is used during the main trials. This keeps the natural stochastic behavior of each planner. The repeated trials are then used to report the mean, standard deviation, and confidence interval of each metric.

B. Map Configuration

Three square environments are used. The map sizes are 10×10, 25×25, and 50×50. The start point is placed near the lower-left corner. The goal point is placed near the upper-right corner. Each trial uses a randomly generated obstacle layout.

The obstacles include circles, rectangles, squares, and triangles. The number of obstacles increases with map size. This makes the larger maps more difficult. All obstacles remain fixed during one planning run. Therefore, the experiments are limited to static environments (Table II).

C. Collision Checking

Collision checking is applied to both nodes and edges. A sampled node is rejected if it lies inside an obstacle or too close to an obstacle boundary. An edge is also rejected if any interpolated point along the edge intersects an obstacle. This prevents the planner from accepting paths that pass through occupied space.

For each edge, points are sampled at a fixed interpolation interval. If any point violates the collision rule, the full edge is discarded. A small collision buffer is used to keep the path away from obstacle boundaries.

D. Baseline Planners

RRT grows a tree from the start point. At each iteration, it samples a point from the map. The nearest tree node is found. The tree is then extended from the nearest node toward the sampled point.

Let q_{near} be the nearest node and q_{sample} be the sampled point. The new node is computed as:

$$q_{\text{new}} = q_{\text{near}} + \min\left(\eta, \left\|q_{\text{sample}} - q_{\text{near}}\right\|_2\right) \frac{q_{\text{sample}} - q_{\text{near}}}{\left\|q_{\text{sample}} - q_{\text{near}}\right\|_2}. \quad (1)$$

Here, η is the step size.

The Euclidean distance between two points $q_i = (x_i, y_i)$ and $q_j = (x_j, y_j)$ is:

$$d(q_i, q_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \quad (2)$$

RRT* extends RRT by adding parent selection and rewiring. For each new node, the algorithm searches nearby nodes. It selects the parent that gives the lowest collision-free cost. It then checks whether nearby nodes can be rewired through the new node to reduce path cost. This improves path quality over time.

E. Proposed N-DQN-RRT Algorithm

The proposed N-DQN-RRT algorithm combines DQN-guided sampling with an RRT*-style tree structure. The method has two phases. The first phase trains a DQN agent. The second phase uses the trained agent to guide sampling during planning.

The planner does not rely only on the DQN. It also keeps random sampling. This reduces the risk of policy bias. It also helps the tree recover when the learned policy gives a poor direction.

1) State representation: The DQN agent receives a seven-dimensional state vector. The state describes the current node, the goal, the nearest obstacle, and the local free space. It is defined as

$$s_t = \left[d_x,\, d_y,\, d_{\text{goal}},\, d_{\text{obs}},\, \delta x_{\text{obs}},\, \delta y_{\text{obs}},\, f_{\text{free}}\right]^T. \quad (3)$$

The terms d_x and d_y are the normalized horizontal and vertical offsets from the current node to the goal. The term d_{goal} is the normalized distance to the goal. The term d_{obs}

is the normalized distance to the nearest obstacle. The terms δx_{obs} and δy_{obs} describe the relative direction of the nearest obstacle. The term f_{free} is the fraction of nearby probe directions that are collision-free.

2) Action space: The action space contains eight discrete directions. The directions are spaced by 45°. Each action selects a direction for guided sampling. The sample distance is drawn from a small random range. This keeps the guided sampler from producing the same path repeatedly.

3) Q-network architecture: The DQN uses a feed-forward neural network. The input layer has seven units. There are two hidden layers with 128 units each. ReLU activation is used in the hidden layers. The output layer has eight units. Each output gives the estimated Q-value of one action. A target network is used during training. The target network has the same architecture as the main Q-network. Its weights are copied from the main network at fixed intervals. This makes the target values more stable during learning.

4) Reward function: The reward function encourages three behaviors. The agent should move toward the goal. It should avoid obstacles. It should avoid unnecessary long movement.

The reward at step t is defined as:

$$r_t = \begin{cases} -10, & \text{if a collision occurs,} \\ +100, & \text{if } d(q_{t+1}, q_{\text{goal}}) \le r_{\text{goal}}, \\ 3\Delta d_{\text{goal}} + 5\Delta c_{\text{obs}} - 0.05, & \text{otherwise.} \end{cases} \tag{4}$$

Here, r_{goal} is the reward threshold near the goal. It is different from the planning goal radius used to stop the algorithm. The term Δd_{goal} is the reduction in distance to the goal:

$$\Delta d_{\text{goal}} = d(q_t, q_{\text{goal}}) - d(q_{t+1}, q_{\text{goal}}). \tag{5}$$

The term Δc_{obs} is the change in obstacle clearance:

$$\Delta c_{\text{obs}} = c(q_{t+1}) - c(q_t). \tag{6}$$

A positive value means that the new position is farther from the nearest obstacle. A negative value means that it is closer to an obstacle.

5) DQN training: The agent is trained with experience replay. Each transition is stored as:

$$(s_t, a_t, r_t, s_{t+1}, done_t). \tag{7}$$

Mini-batches are sampled from the replay buffer during training. The Q-network is trained by minimizing the mean squared Bellman error:

$$L(\theta) = \mathbb{E}\left[\left(y_t - Q(s_t, a_t; \theta)\right)^2\right]. \tag{8}$$

The target value is

$$y_t = r_t + \gamma(1 - done_t) \max_{a'} Q(s_{t+1}, a'; \theta^-). \tag{9}$$

Here, θ denotes the main network weights. The term θ^- denotes the target network weights. The discount factor is γ. The term $done_t$ is 1 when the episode ends and 0 otherwise.

The agent uses an epsilon-greedy policy during training. At the beginning, the agent explores more. As training continues, exploration is reduced. The epsilon value is updated as:

$$\epsilon_{k+1} = \max(\epsilon_{\min}, \beta\epsilon_k). \tag{10}$$

Here, $\epsilon_{\min}$ is the minimum exploration rate. The term β is the decay factor.

6) Adaptive rewiring radius: N-DQN-RRT uses an adaptive rewiring radius. The radius changes according to local obstacle information. It becomes smaller near cluttered regions. It becomes larger in open regions.

The adaptive radius is computed as:

$$r_{\text{adaptive}} = \text{clip}\left(r_{\text{base}}\left(0.5 + 0.8d_{\text{obs}} + 0.7f_{\text{free}}\right), 0.6r_{\text{base}}, 2.0r_{\text{base}}\right). \tag{11}$$

Here, r_{base} is the base rewiring radius. The term d_{obs} is the normalized obstacle distance. The term f_{free} is the local free-space fraction. The function $\text{clip}(\cdot)$ keeps the radius within the lower and upper bounds.

7) Planning phase: During planning, the tree starts at the start node. At each iteration, a candidate sample is generated in one of three ways. A small portion of samples is placed at the goal. Another portion is sampled uniformly at random. The remaining samples are generated through the trained DQN policy.

The sampling design follows an 80/20 directed-random strategy. In this design, 75% of samples are generated by the DQN policy and 5% are placed at the goal as goal bias. The remaining 20% are sampled uniformly at random. Thus, 80% of the samples are directed and 20% preserve global exploration.

The sampling rule is:

$$q_{\text{sample}} = \begin{cases} q_{\text{goal}}, & \text{with probability } 0.05, \\ q_{\text{rand}}, & \text{with probability } 0.20, \\ q_{\text{DQN}}, & \text{with probability } 0.75. \end{cases} \tag{12}$$

After a sample is generated, the nearest node is found. A new node is created using Eq. (1). The node is added only if the point and edge are collision-free. The best parent is selected from nearby nodes. Then nearby nodes are rewired if the new connection lowers their path cost.

The planning process stops when the tree reaches the goal region. The final path is extracted by tracing parent links from the goal node back to the root.

F. Hyperparameter Settings

The training and planning hyperparameters of N-DQN-RRT are presented in Table III.

TABLE III. N-DQN-RRT HYPERPARAMETERS

Hyperparameter	Value
Training episodes	400
Maximum steps per episode	300
Batch size	64
Learning rate	0.001
Optimizer	Adam
Discount factor γ	0.95
Initial epsilon ϵ_0	1.0
Epsilon decay β	0.997
Minimum epsilon during training	0.05
Planning epsilon	0.02
Replay buffer capacity	8000
Hidden layer size	128×128
Network structure	$7 \rightarrow 128 \rightarrow 128 \rightarrow 8$
Target update frequency	50 gradient steps
Collision buffer	0.05 units
Edge interpolation step	0.02 units
Base rewiring radius	2.0 units

G. Algorithm Summary

Algorithm 1 presents the pseudocode of the proposed N-DQN-RRT algorithm. Fig. 1 shows the corresponding flowchart. The algorithm starts from the root node. It then expands the tree until the goal region is reached or the maximum number of iterations is exceeded.

Algorithm 1 N-DQN-RRT

Require: Start node q_{start}, goal node q_{goal}, obstacle set, maximum iterations, trained DQN agent
Ensure: Collision-free path or failure

```
Initialize tree T with root node q_start
for i = 1 to maximum iterations do
    Generate q_sample using Eq. (12)
    Find nearest node q_near in T
    Compute q_new using Eq. (1)
    if q_new is in collision or edge (q_near, q_new) is in collision then
        Continue
    end if
    Compute r_adaptive using Eq. (11)
    Find neighbor nodes within r_adaptive
    Select the lowest-cost collision-free parent
    Add q_new to T
    Rewire nearby nodes if the new connection reduces cost
    if d(q_new, q_goal) is within the goal radius then
        Add goal node
        return extracted path
    end if
end for
return failure
```

Fig. 1 shows the main steps of N-DQN-RRT. The tree is first initialized at the start node. At each iteration, a candidate sample is generated using goal bias, random sampling, or the trained DQN policy. The candidate node and edge are then checked for collision. If the candidate is valid, the planner selects a parent, adds the new node, and applies adaptive rewiring. The process stops when the goal region is reached. If the maximum number of iterations is reached first, the planner returns failure.

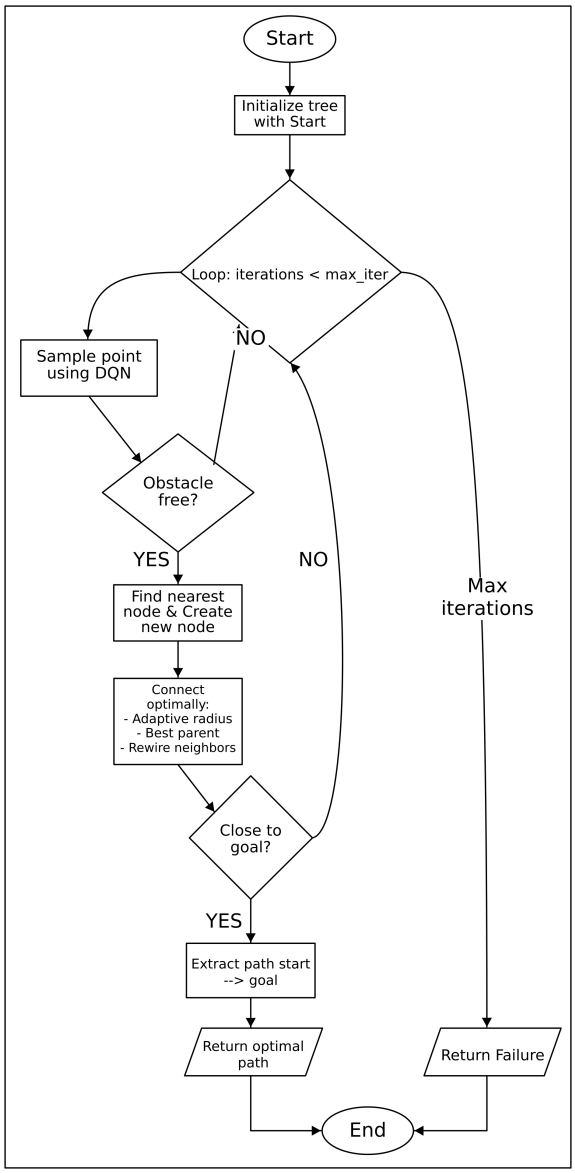

Fig. 1. Flowchart of the proposed N-DQN-RRT path planning algorithm.

H. Evaluation Metrics

Six metrics are used to evaluate the planners. Path length measures the total Euclidean length of the final path. Steering angle measures the average turning angle between consecutive path segments. Iterations measure the number of tree expansion steps. Sampling size measures the number of nodes in the final tree. Computation time measures wall-clock planning time. Success rate measures the percentage of trials that reach the goal within the iteration limit.

For each metric, the mean and standard deviation are

reported across 100 trials. A 95% confidence interval is also computed for the main continuous metrics. Path length is calculated only for successful trials. Success rate is reported over all trials.

The 95% confidence interval is computed as

$$CI = \bar{x} \pm t_{\alpha/2,n-1}\frac{s}{\sqrt{n}}, \tag{13}$$

where, $\bar{x}$ is the sample mean. The term s is the sample standard deviation. The term n is the number of trials. The term $t_{\alpha/2,n-1}$ is the critical value from the t distribution.

I. Reproducibility Details

The implementation uses a three-layer Q-network with input size 7 and output size 8. All state features are normalized before being passed to the network. The optimizer is Adam with a learning rate of 0.001. The loss function is the mean squared Bellman error. The collision buffer is set to 0.05 units. The edge interpolation step is set to 0.02 units. The base rewiring radius is set to 2.0 units.

The main experiments use 100 independent trials per map. This supports statistical reporting under stochastic planning. For exact reproducibility, the final implementation should also report the obstacle generation rules, software versions, hardware configuration, and random seed list.

IV. Results and Discussion

This section presents the results of RRT, RRT*, and the proposed N-DQN-RRT. The algorithms were tested on three static two-dimensional maps. The map sizes were 10×10, 25×25, and 50×50. Each map setting was evaluated using 100 independent trials. The start and goal positions were fixed for each map. The obstacle layout was randomly generated in each trial.

Six metrics were used for comparison. These were path length, steering angle, iteration count, sampling size, computation time, and success rate. Path length, steering angle, iteration count, sampling size, and computation time were computed from successful trials. Success rate was computed over all 100 trials.

A. Map 1 Results: 10×10 Environment

The first experiment used a 10×10 environment. The start position was $(1.0, 1.0)$. The goal position was $(9.0, 9.0)$. The step size was 1.0. The maximum number of iterations was 250 for all algorithms (Table IV).

All three algorithms reached the goal in every trial. This shows that the smallest map was manageable for all planners under the given iteration limit. However, N-DQN-RRT produced better paths and used fewer samples.

Compared with RRT*, N-DQN-RRT reduced the mean path length from 13.90 m to 12.50 m. This is a reduction of about 10.1%. It also reduced the mean steering angle from 40.07° to 22.63°. This indicates a smoother path. The iteration count dropped from 79 to 24. The sampling size also dropped from 61 to 22.

The confidence intervals for path length do not overlap between N-DQN-RRT and RRT*. This supports a meaningful difference in the tested trials. It should not be treated as mathematical proof. It shows that the observed improvement is unlikely to be caused only by random variation in these experiments.

B. Map 2 Results: 25×25 Environment

The second experiment used a 25×25 environment. The start position was $(1.0, 1.0)$. The goal position was $(24.0, 24.0)$. The step size was 1.0. The maximum number of iterations was 500 (Table V).

The second map shows a clear difference between the planners. RRT reached the goal in 28% of the trials. RRT* reached the goal in 38% of the trials. N-DQN-RRT reached the goal in all trials. This is an increase of 62 percentage points over RRT*.

Among successful trials, N-DQN-RRT also produced shorter paths. Its mean path length was 36.60 m. The mean path length of RRT* was 41.10 m. This is a reduction of about 10.9%. The mean steering angle was also much lower. This suggests that the learned sampling policy helped produce smoother routes.

N-DQN-RRT required fewer iterations and fewer nodes than RRT*. The iteration count decreased from 404 to 186. The sampling size decreased from 263 to 103. The computation time also decreased from 0.60 s to 0.11 s. This reduction is mainly due to the smaller tree size and fewer expansion steps.

The standard deviation of the N-DQN-RRT iteration count is still high. This shows that some random obstacle layouts were harder than others. Even so, the method reached the goal in every trial.

C. Map 3 Results: 50×50 Environment

The third experiment used the largest environment. The map size was 50×50. The start position was $(1.0, 1.0)$. The goal position was $(49.0, 49.0)$. The step size was 1.0. The maximum number of iterations was 1500 (Table VI).

The 50×50 map was the most difficult case. RRT succeeded in 10% of the trials. RRT* succeeded in 14% of the trials. N-DQN-RRT succeeded in all 100 trials. This is an increase of 86 percentage points over RRT*.

The path length also improved. N-DQN-RRT achieved a mean path length of 76.80 m. RRT* achieved 87.20 m. This is a reduction of about 11.9%. The steering angle was reduced from 33.76° to 17.04°. This again shows smoother path behavior.

The largest gain was seen in computation time and sampling size. N-DQN-RRT used 238 nodes on average. RRT* used 955 nodes on average. The computation time decreased from 6.39 s to 0.52 s. This suggests that the DQN-guided sampler reduced unnecessary tree growth in the tested maps.

The standard deviation of N-DQN-RRT computation time is high. This means that a few trials were much harder than the average case. The method still reached the goal in every run.

TABLE IV. MAP 1 RESULTS OVER 100 TRIALS IN THE 10×10 ENVIRONMENT

Metric	RRT	RRT*	N-DQN-RRT
Path length	14.60 ± 1.40 m	13.90 ± 1.06 m	$\mathbf{12.50 \pm 0.83}$ m
Iterations	69 ± 24	79 ± 40	$\mathbf{24 \pm 10}$
Sampling size	55 ± 17	61 ± 26	$\mathbf{22 \pm 5}$
Steering angle	$42.67^\circ \pm 8.74^\circ$	$40.07^\circ \pm 8.26^\circ$	$\mathbf{22.63^\circ \pm 7.83^\circ}$
Computation time	0.02 ± 0.01 s	0.04 ± 0.05 s	$\mathbf{0.01 \pm 0.004}$ s
Success rate	100.0%	100.0%	**100.0%**
95% CI for path length	[14.34, 14.89]	[13.68, 14.10]	$[\mathbf{12.32}, \mathbf{12.65}]$
95% CI for computation time	[0.0141, 0.0188]	[0.0300, 0.0500]	$[\mathbf{0.0074}, \mathbf{0.0087}]$

TABLE V. MAP 2 RESULTS OVER 100 TRIALS IN THE 25×25 ENVIRONMENT

Metric	RRT	RRT*	N-DQN-RRT
Path length	42.10 ± 2.83 m	41.10 ± 2.14 m	$\mathbf{36.60 \pm 1.87}$ m
Iterations	407 ± 67	404 ± 62	$\mathbf{186 \pm 95}$
Sampling size	270 ± 46	263 ± 44	$\mathbf{103 \pm 34}$
Steering angle	$36.79^\circ \pm 4.91^\circ$	$37.64^\circ \pm 4.38^\circ$	$\mathbf{17.82^\circ \pm 3.43^\circ}$
Computation time	0.34 ± 0.12 s	0.60 ± 0.27 s	$\mathbf{0.11 \pm 0.06}$ s
Success rate	28.0%	38.0%	**100.0%**
95% CI for path length	[41.05, 43.24]	[40.39, 41.79]	$[\mathbf{36.18}, \mathbf{36.93}]$
95% CI for computation time	[0.2938, 0.3891]	[0.5173, 0.6925]	$[\mathbf{0.0962}, \mathbf{0.1194}]$

TABLE VI. MAP 3 RESULTS OVER 100 TRIALS IN THE 50×50 ENVIRONMENT

Metric	RRT	RRT*	N-DQN-RRT
Path length	89.70 ± 3.41 m	87.20 ± 3.95 m	$\mathbf{76.80 \pm 3.11}$ m
Iterations	1259 ± 232	1345 ± 109	$\mathbf{317 \pm 203}$
Sampling size	901 ± 145	955 ± 86	$\mathbf{238 \pm 135}$
Steering angle	$35.92^\circ \pm 3.64^\circ$	$33.76^\circ \pm 2.94^\circ$	$\mathbf{17.04^\circ \pm 2.24^\circ}$
Computation time	3.56 ± 1.30 s	6.39 ± 1.33 s	$\mathbf{0.52 \pm 0.46}$ s
Success rate	10.0%	14.0%	**100.0%**
95% CI for path length	[87.23, 92.10]	[84.88, 89.45]	$[\mathbf{76.21}, \mathbf{77.45}]$
95% CI for computation time	[2.6353, 4.4926]	[5.6176, 7.1531]	$[\mathbf{0.4273}, \mathbf{0.6085}]$

This supports its practical stability under the tested random obstacle settings.

It is important to interpret the results carefully. For Maps 2 and 3, RRT and RRT* failed in many trials. Their path length and computation time values are based only on successful trials. These successful cases may represent easier obstacle layouts. Therefore, success rate is the most important metric for larger maps.

D. Cross-Environment Analysis

Table VII summarizes the performance of N-DQN-RRT across the three map sizes. The method kept a 100% success rate in all tested environments. The path length increased as the map size increased. This is expected because the start and goal were farther apart in larger maps.

The steering angle decreased from Map 1 to Map 3. This may be because larger maps provide more space for gradual movement. It may also reflect the effect of DQN-guided directional sampling. Further testing is needed before making a stronger claim.

The iteration count and sampling size increased with map size. This is expected because larger maps require longer paths and more expansion steps. However, the increase was still lower than the growth seen in RRT* computation time. In these experiments, N-DQN-RRT increased from 0.01 s to 0.52 s. RRT* increased from 0.04 s to 6.39 s. This indicates better empirical scaling under the tested settings.

E. Discussion

The results show that the proposed N-DQN-RRT planner improves practical planning performance in the tested static two-dimensional environments. The improvement is most visible in the larger maps. In those maps, RRT and RRT* often failed to reach the goal within the iteration limit. N-DQN-RRT reached the goal in every trial.

The main reason is the sampling strategy. The DQN policy guides most samples toward useful directions. The random sampling component keeps some global exploration. This helps reduce the risk of relying only on the learned policy. The goal bias also helps the planner converge near the target region.

The adaptive rewiring radius also contributes to path quality. In open regions, a larger radius allows the planner to consider more possible parent nodes. Near obstacles, a smaller radius reduces unnecessary neighbor checks. This can reduce collision-heavy rewiring attempts. It can also help produce smoother local connections.

The results should not be overinterpreted. The experiments do not prove probabilistic completeness or asymptotic optimality for N-DQN-RRT. They only show empirical improvement under the tested simulation conditions. The training and testing

TABLE VII. N-DQN-RRT PERFORMANCE ACROSS MAP CONFIGURATIONS

Metric	**Map 1:** 10×10	**Map 2:** 25×25	**Map 3:** 50×50
Success rate	100%	100%	100%
Path length	12.50 ± 0.83 m	36.60 ± 1.87 m	76.80 ± 3.11 m
Steering angle	$22.63^\circ \pm 7.83^\circ$	$17.82^\circ \pm 3.43^\circ$	$17.04^\circ \pm 2.24^\circ$
Iterations	24 ± 10	186 ± 95	317 ± 203
Sampling size	22 ± 5	103 ± 34	238 ± 135
Computation time	0.010 ± 0.004 s	0.110 ± 0.060 s	0.520 ± 0.460 s

maps were generated from the same general distribution. This means that performance on different map types is still uncertain. Examples include narrow passages, maze-like maps, dynamic obstacles, and real robot environments.

There are also reproducibility limits. The trials were repeated 100 times, but no fixed random seed list was used. This supports statistical reporting, but it does not provide exact run-to-run reproduction. Future experiments should include fixed seed lists, obstacle-generation details, and code release. Additional baselines such as Informed RRT*, Bi-RRT*, and goal-biased RRT* would also make the comparison stronger.

Overall, the results support the use of DQN-guided sampling as a practical improvement for RRT*-style planning in static 2D environments. The strongest evidence is the increase in success rate and the reduction in sampling size on larger maps. The method is promising, but it should be tested further before being claimed as general or deployment-ready.

V. CONCLUSION

This paper proposed and evaluated N-DQN-RRT, a hybrid path-planning algorithm for static two-dimensional environments. The method integrates a trained Deep Q-Network agent into an RRT*-style planning framework. It uses a seven-dimensional state representation that includes goal direction, obstacle distance, obstacle direction, and local free-space information. It also uses an 80/20 directed-random sampling strategy. In this strategy, 75% of samples are guided by the DQN policy, 5% are assigned to goal bias, and 20% are sampled uniformly at random. An adaptive rewiring radius is also used to adjust the local optimization range according to obstacle conditions.

The proposed method was implemented in Python and tested in three static two-dimensional random obstacle environments. The map sizes were 10×10, 25×25, and 50×50. Each map setting was evaluated using 100 independent stochastic trials. The performance of N-DQN-RRT was compared with classical RRT and RRT* using path length, steering angle, iteration count, sampling size, computation time, and success rate.

The results show that N-DQN-RRT improved empirical planning performance under the tested conditions. It achieved a 100% success rate across all 300 trials. In comparison, RRT achieved 28–100% success and RRT* achieved 38–100% success depending on the map size. In the largest 50×50 environment, N-DQN-RRT reduced path length by 11.9%, steering angle by 49.5%, sampling size by 75.1%, and computation time by 91.9% compared with RRT*. The non-overlapping 95% confidence intervals support the observed differences in the tested trials.

The main contribution of this work is empirical and architectural. The paper does not claim to be the first study on DQN-RRT integration. Instead, it studies a specific hybrid design that combines DQN-guided sampling, goal bias, random exploration, and adaptive rewiring within an RRT*-style planner. The results suggest that learned sampling can reduce unnecessary exploration and improve practical planning efficiency in static two-dimensional random obstacle environments.

This study has several limitations. First, the evaluation is limited to static two-dimensional maps. Dynamic obstacles, moving agents, and real-world sensing uncertainty are not considered. Second, the DQN agent is trained and tested on maps generated from the same general distribution. This limits the strength of the generalization claim. Third, no physical robot experiment is included. The simulation does not model sensor noise, actuator delay, localization error, or communication latency.

Future work should extend N-DQN-RRT to dynamic environments that require online replanning. The method should also be tested on physical mobile robots using sensors such as LiDAR or depth cameras. Further experiments should include three-dimensional planning tasks for UAVs and robotic manipulators. Ablation studies are also needed to measure the contribution of each component. These studies should test different DQN-random sampling ratios, remove adaptive rewiring, and compare fixed-radius and adaptive-radius variants. Cross-map generalization tests should also be added by training on one obstacle distribution and testing on different map structures such as narrow passages and maze-like environments.

REFERENCES

[1] O. Al-Sanjary *et al.*, "A hybrid method of enhancing accuracy of facial recognition system using gabor filter and stacked sparse autoencoders deep neural network," *Applied Sciences*, vol. 12, no. 21, p. 11052, 2022.

[2] L. Yang *et al.*, "Path planning technique for mobile robots: A review," *Machines*, vol. 11, no. 10, p. 980, 2023.

[3] A. N. A. Rafai, N. Adzhar, and N. I. Jaini, "A review on path planning and obstacle avoidance algorithms for autonomous mobile robots," *Journal of Robotics*, vol. 2022, p. 2538220, 2022.

[4] R. Agrawal, B. Singh, and R. Kumar, "Classical approaches for mobile robot path planning: A review," in *Proceedings of the 3rd IEEE International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, 2022, pp. 400–405.

[5] L. B. P. Nascimento, D. Barrios-Aranibar, V. G. Santos, D. S. Pereira, W. C. Ribeiro, and P. J. Alsina, "Safe path planning algorithms for mobile robots based on probabilistic foam," *Sensors*, vol. 21, no. 12, p. 4156, 2021.

[6] P. Xin, X. Wang, X. Liu, Y. Wang, Z. Zhai, and X. Ma, "Improved bidirectional rrt* algorithm for robot path planning," *Sensors*, vol. 23, no. 2, p. 1041, 2023.

[7] S. Ganesan, B. Ramalingam, and R. E. Mohan, "A hybrid sampling-based rrt* path planning algorithm for autonomous mobile robot navigation," *Expert Systems with Applications*, vol. 258, p. 125206, 2024.

[8] T. Zhang, J. Wang, and M. Q.-H. Meng, "Generative adversarial network based heuristics for sampling-based path planning," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 1, pp. 64–74, 2022.

[9] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural rrt*: Learning-based optimal path planning," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 1748–1758, 2020.

[10] L. Xu and W. Zhang, "Survey on path planning based on deep reinforcement learning," in *Proceedings of the 2025 2nd International Conference on Machine Learning and Intelligent Computing*, ser. Proceedings of Machine Learning Research, vol. 278. PMLR, 2025, pp. 685–695. [Online]. Available: https://proceedings.mlr.press/v278/xu25a.html

[11] H. T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[12] H. Liu, Y. Gu, X. Li, and X. Xiao, "Deep reinforcement learning integrated rrt algorithm for path planning," in *Proceedings of the 2023 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, 2023, pp. 239–244.

[13] A. Alexander, K. Venkatesan, J. Mounsef, and K. Ramanujam, "A comprehensive survey of path planning algorithms for autonomous systems and mobile robots: Traditional and modern approaches," *IEEE Access*, 2025.

[14] A. Muhammad, N. R. H. Abdullah, M. A. H. Ali, I. H. Shanono, and R. Samad, "Simulation performance comparison of a*, gls, rrt and prm path planning algorithms," in *Proceedings of the 12th IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, 2022, pp. 258–263.

[15] J. Wang, X. Jia, T. Zhang, N. Ma, and M. Q.-H. Meng, "Deep neural network enhanced sampling-based path planning in 3d space," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3434–3443, 2022.

[16] L. Yang, J. Bi, and H. Yuan, "Dynamic path planning for mobile robots with deep reinforcement learning," in *IFAC-PapersOnLine*, 2022, pp. 19–24.

[17] Y. Li, "An intelligent vehicle path planning algorithm based on an improved dqn for use in complex environments," in *Proceedings of the 7th International Conference on Internet of Things, Automation and Artificial Intelligence (IoTAAI)*, 2025, pp. 628–632.

[18] X. Jin and Z. Wang, "Proximal policy optimization based dynamic path planning algorithm for mobile robots," *Electronics Letters*, 2022.

[19] Y. Zhang and P. Chen, "Path planning of a mobile robot for a dynamic indoor environment based on an sac-lstm algorithm," *Sensors*, vol. 23, no. 24, p. 9802, 2023.

[20] P. Li, D. Chen, Y. Wang, L. Zhang, and S. Zhao, "Path planning of mobile robot based on improved td3 algorithm in dynamic environment," *Heliyon*, vol. 10, no. 11, p. e32167, 2024.

[21] H.-L. Pham, N.-N. Bui, and T.-V. Dang, "Hybrid path planning for wheeled mobile robot based on rrt-star algorithm and reinforcement learning method," *Journal of Robotics and Control (JRC)*, vol. 6, no. 4, pp. 2045–2051, 2025.

[22] X. Xu, Y. Cao, and X. Liu, "Improved dqn algorithm for path planning of autonomous mobile robots," 2023, research Square preprint.