Extended Abstract

Motivation While path planning has been extensively studied, efficient coverage-based planning in complex, cluttered indoor environments remains a relatively underexplored area, particularly in the context of cleaning robots. Most existing reinforcement learning approaches focus on navigating to a single goal or improving path efficiency within known environments. However, in real-world domestic settings, robots must adapt to previously unseen layouts, minimize redundant motion, and maximize spatial coverage without retraining for each new environment. This project aims to train models that can efficiently explore an unseen environment and cover all grids that need to be cleaned.

Method We trained the PPO and DQN models through customized reward shaping after observing agent training behaviors. Multiple encoding schemes, such as encoding the position and environment knowledge via a masked multi-dimensional array and pre-processing through convolutional layers, are employed to better encode spatial knowledge. Rainbow DQN, an enhanced variant of DQN, integrates six extensions: double Q-learning, dueling network architecture, n-step bootstrapping, prioritized experience replay, distributional Q-learning, and noisy linear layers for exploration. Unlike standard DQN, Rainbow replaces the TD error with a Kullback–Leibler divergence between distributions over returns, using multi-step targets and categorical atoms projected onto a fixed support. We trained and tested Rainbow DQN also using customized reward shaping and tuned hyperparameters for all algorithms.

Implementation & Results We model the household environment as a rectangular grid space and the vacuum robot as a single grid unit. The robot can take one of three actions at each time step: rotate left by 45 degrees, rotate right by 45 degrees, or move forward. The base environment defines reward at each timestep and at task completion. We implemented three algorithms to solve this problem: PPO, DQN, and Rainbow DQN. Each model has its own on-policy reward shaping wrappers to resolve its own challenges. Early stopping is used to prevent prolonged episodes. The observation space is flattened and normalized for more stable training behavior, and the hyperparameters are tuned with `Optuna`. We implemented an agent that takes random actions at each step as our baseline.

All of the three algorithms are trained on multiple environments: a 20×20 grid with no walls and all grids need to be covered; a 20×20 grid with random walls separating the space into 4 connected rooms and all grids need to be covered; a 20×20 grid with random walls and with 5 random dirt clusters of particular shapes; a 40×30 grid with realistic studio layout and all grids need to be covered. The algorithms are evaluated on separate evaluation layouts that mimic the training setup for each environment difficulty.

We found that PPO consistently achieved the highest coverage and most structured trajectories, particularly in environments with clear reward signals and spatial structure. Its on-policy updates and stability enabled reliable generalization, even in cluttered or large layouts. In contrast, DQN performed reliably in simpler environments with open spaces but often failed to generalize in larger or more structured maps, getting stuck or terminating prematurely. Rainbow DQN showed potential in moderate environments, benefiting from prioritized replay and distributional updates, but was sensitive to reward design and often unstable in large, cluttered maps. Overall, policy-gradient methods like PPO proved more robust when reward shaping aligned with the multi-phase nature of the task, while value-based methods revealed trade-offs between architectural complexity and learning robustness.

Discussion & Conclusion Throughout the project, we found that reinforcement learning in spatial cleaning tasks is highly sensitive to both reward structure and observation encoding. Reward shaping emerged as a critical design choice, particularly in distinguishing the phases of exploration, coverage, and goal completion. Carefully designed rewards significantly improved performance across all algorithms compared to learning from base rewards alone. Our experiments also underscore the importance of spatial information encoding. Using one-hot position maps and orientation vectors helped preserve locality, but flattening grid layouts (often done to improve numerical stability) compromised spatial structure unless convolutional layers were applied. However, computational constraints limited extensive tuning of convolutional neural network-based models. Hyperparameter optimization with `Optuna` added structure to training but had mixed effectiveness: it yielded local improvements while sometimes failing to discover robust global configurations, especially in high-variance, stochastic settings like this one. Stability was improved through multi-seed trials, albeit at a significant computational cost. Altogether, these findings illustrate the nuanced trade-offs involved in applying deep reinforcement learning to structured spatial domains and highlight promising directions in architectural design and reward curriculum development.

Reinforcement Learning for Covered Path Planning on Robot Vacuum

Ngoc Vo
Department of Statistics
Stanford University
hnngocvo@stanford.edu

Yiran Fan
Department of Statistics
Stanford University
yiranf@stanford.edu

Siqi Ma
Department of Statistics
Stanford University
siqima@stanford.edu

Abstract

This project explores efficient coverage-based planning for cleaning robots using deep reinforcement learning in cluttered, unseen indoor environments. We implement PPO, DQN, and Rainbow DQN with customized reward shaping and spatial encoding (e.g., masked maps, orientation vectors) to improve navigation and cleaning performance. The agents are trained and evaluated across diverse grid-based environments with varying layouts and dirt distributions. PPO consistently achieves the most reliable and complete coverage, while DQN performs well in simple maps but struggles with complexity. Rainbow DQN offers potential through distributional updates and prioritized replay, but suffers from instability. Our results highlight the importance of reward phase design and spatial structure in DRL, while reflecting broader challenges in tuning and generalization across high-variance, multi-objective tasks.

1 Introduction

While path planning problems have been extensively studied, most existing works focus on navigating through a complex environment, and efficient covered planning in complex, cluttered indoor environments remains a relatively underexplored area, particularly in the context of cleaning robots. Most existing reinforcement learning approaches focus on navigating to a single goal or improving path efficiency within known environments. However, in real-world domestic settings, robots must adapt to previously unseen layouts, minimize redundant motion, and maximize spatial coverage without retraining for each new environment.

This project proposes a reinforcement learning (RL) framework for intelligent, coverage-aware path planning tailored to cleaning robots. Unlike conventional deep reinforcement learning (DRL) applications that optimize for point-to-point navigation, our approach seeks to identify paths that maximize area coverage while minimizing overlap and energy usage. Through this work, we aim to advance RL-based path planning toward practical, real-world deployment in domestic robotics, where adaptability, efficiency, and robustness are essential.

2 Related Work

Conventional path planning methods like A*, Dijkstra, and sampling-based approaches, such as Rapidly-exploring Random Trees, often rely on explicit environmental representations, which limit their adaptability to new or partially observable settings. In contrast, RL offers a model that learns optimal policies through interaction, enabling agents to make informed decisions based solely on sensor inputs.

Recent work by Moon et al. (2022) focuses specifically on intelligent path planning for cleaning robots using Proximal Policy Optimization (PPO). Their approach combines PPO with techniques such as transfer learning, detection of the nearest uncleaned tile, reward shaping, and elite sampling to enhance generalization across various indoor environments. Other approaches, such as those by Qin et al. (2022) and Quiñones-Ramírez et al. (2023), apply PPO and DQN variants for navigation using only local observations. These studies demonstrate effective obstacle avoidance and goal-reaching in partially observable 2D environments.

However, their primary objective remains optimizing efficiency in a known or retrained environment, rather than achieving generalization without retraining. Additionally, while the method shows superiority over random and zigzag baselines, its evaluation metrics emphasize task efficiency and convergence speed rather than total spatial coverage.

Qin et al. (2022) similarly apply PPO to path planning in unknown 2D environments where only local information (e.g., from 2D LiDAR) is available. They define the problem as a Partially Observed Markov Decision Process (POMDP) and show that effective policies can be learned using local sensing without a global map. This work prioritizes real-time feasibility and safety in navigation. However, the goal remains single-point destination reaching rather than area-wide exploration or maximizing coverage.

Quiñones-Ramírez et al. (2023) extend DRL applications in 2D environments using Deep Q-Network (DQN) variants, such as D3QN and Rainbow, for tasks like obstacle avoidance and goal-reaching. A key contribution of their study is an in-depth exploration of how reward function design affects learned behaviors. Nonetheless, the focus remains on navigating to a single goal, without explicit attention to maximizing area coverage or adapting across diverse home layouts.

More recently, research has begun addressing coverage path planning (CPP) directly. Jonnarth et al. (2024) propose a Soft Actor-Critic (SAC)-based framework that leverages multi-scale egocentric maps and introduces a novel total variation-based reward to encourage uniform spatial coverage in unknown environments. Carvalho and Aguiar (2025) further advance this direction with a zero-shot DRL framework that generalizes CPP policies across unseen layouts without retraining, using action masking and structured observation embeddings.

These efforts highlight the growing interest in coverage-aware DRL, yet most existing approaches rely on structured, idealized environments that overlook real-world constraints. Our work builds on this direction by exploring a broader set of DRL algorithms aimed at maximizing spatial coverage, while introducing energy-aware objectives that explicitly penalize redundancy and account for realistic domestic challenges, such as flexible room geometries, obstacles, surface restrictions, and power limitations. Through these contributions, we seek to advance DRL-based coverage planning toward more practical and deployable solutions.

3 Method

3.1 Environment Setup

States The cleaning task is modeled as a discrete 2D grid environment, representing a simplified household layout. Each cell in the grid is initialized to one of the three states: clean, dirty, or obstacle. The robot is initialized at a known, non-obstacle location, adjacent to a wall. The primary task objective is to maximize the total dirty area covered by the agent, where each cleanable cell provides an equal reward upon first visit. The environment is designed to be flexible in size, the wall or obstacle setup, and the dirty area layout.

Action Space The agent interacts with the environment via a discrete action space. It can move forward in the direction it is currently facing, stay in place, or rotate in place by 45-degrees. This configuration supports both fine maneuvering required to traverse the floor plans.

Observation Space The agent has access to its starting position, current position, dirt map, memory map of explored obstacles, all of which have the same size as the 2D grid environment, encoded as binary value vectors. The agent also know its own orientation encoded as an integer from 0 to 7, and a local observation of the 8 surrounding grids.

Transitions At each time step, the agent takes an action first, receiving a reward accordingly. After the action, the agent observes the surrounding 8 grids and updates its memory of the obstacle map. If all grids are cleaned, the agent will receive a large reward. If the agent cleans all necessary grids and returns to the starting position, the agent will receive a larger reward, and the episode is terminated.

3.2 Objective Function and Reward Design

The main objective is learning a policy π that maximizes the expected sum of discounted rewards over time, where the rewards are primarily acquired through covering the entire unvisited space. We allow the agent to achieve an energy-efficient behavior by embedding the costs of movement into the reward function.

To guide the learning process toward coverage-maximizing and energy-efficient behavior, we define a dense reward function with intermediate and terminal components, detailed in Table 1.

Observing the evaluation rollout, we further dynamically provide additional reward shaping wrappers for desired and undesired behaviors. We tailor wrappers for different algorithms as they exhibit different learning behavior. We detailed the wrappers in Table 3, Table 4, Table 5 in the Appendix.

3.3 Algorithms

This project plans to implement and evaluate the following DRL algorithms for finding energy-efficient 2D space coverage paths: Proximal Policy Optimization (PPO), Deep Q-Network (DQN), and Rainbow DQN. We implemented PPO and DQN by

Reward	Type	Reward Design
$+1.0$	Repetitive	Rewawrd for cleaning a dirty grid
-0.5	Repetitive	Discourage invalid actions (i.e.moving into obstacles or out of bounds)
-0.05	Repetitive	Energy consumption for moving forward
-0.001	Repetitive	Energy consumption for rotation
$+5,000.0$	One Time	Reward for completing coverage of all dirty grids
$+10,000.0$	One Time	Reward for complete coverage and return to start position

Table 1: Base environment reward design.

leveraging `stable-baseline-3` with customized evaluation call-backs. Rainbow DQN is implemented from scratch, with additional components from the original paper (Hessel et al., 2017).

Actor-Critic (AC) method combines the advantage of value-based and policy-based approaches, where we fit models to estimate V^π, and take gradient step using an advantage function $A^\pi(s_{i,t}, a_{i,t})$ defined as

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})A^\pi(s_{i,t})$$
$$A^\pi(s_{i,t}, a_{i,t}) \approx r(s_t, a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

with discount factor γ. The value function V^π is estimated through bootstrap or Monte Carlo simulation. PPO improves on AC via a clipped surrogate objective and estimated value function with a varying horizon to achieve better policy updates.

DQN approximates the optimal Q-value function using a deep neural network and updates through policy iteration. However, standard DQN often suffers from overestimation bias and unstable learning due to correlated updates. To address these challenges, several extensions have been proposed: Double DQN introduces a target network to decouple action selection from evaluation; Dueling DQN separates the estimation of state value and advantage functions; and Rainbow DQN combines multiple such improvements into a single unified architecture.

3.3.1 Rainbow DQN

The Rainbow DQN algorithm was proposed by Hessel et al. (2017) to address limitations in the original DQN by integrating several orthogonal enhancements into a single, unified framework. Rather than developing a new algorithm from scratch, the authors empirically studied how six previously proposed DQN extensions—each targeting a distinct aspect of reinforcement learning, could be combined. These include improvements in target estimation (Double DQN), architecture design (Dueling Networks), return computation (Multi-step learning), data sampling (Prioritized Replay), value representation (Distributional RL), and exploration (Noisy Nets). Their integration led to significantly improved performance and sample efficiency across the Atari 2600 benchmark.

Rainbow replaces the standard DQN loss

$$\mathcal{L}_{\text{DQN}}(\theta) = \Big(R_{t+1} + \gamma \max_{a'} Q_\theta(S_{t+1}, a') - Q_\theta(S_t, A_t)\Big)^2 \tag{1}$$

with a multi-step distributional variant, using truncated n-step returns

$$R_t^{(n)} = \sum_{k=0}^{n-1}\gamma^k R_{t+k+1}, \tag{2}$$

and estimating a distribution over returns with categorical atoms. The target distribution is constructed as

$$\mathbf{d}_t^{(n)} = \Big(R_t^{(n)} + \gamma^n \mathbf{z},\ \mathbf{p}_\theta(S_{t+n}, a^*)\Big), \quad \text{where} \quad a^* = \arg\max_a \mathbb{E}_{\mathbf{z}\sim\mathbf{p}_\theta(S_{t+n},a)}[\mathbf{z}], \tag{3}$$

and the loss is the Kullback–Leibler divergence between the projected target and current distribution:

$$\mathcal{L}_{\text{Rainbow}}(\theta) = D_{\text{KL}}\left(\Phi_{\mathbf{z}}\mathbf{d}_t^{(n)} \,\|\, \mathbf{p}_\theta(S_t, A_t)\right). \tag{4}$$

This formulation allows Rainbow to capture return uncertainty, leverage more informative bootstrapping, and maintain stable learning dynamics via architectural and sampling enhancements.

Our implementation of Rainbow DQN integrates six key extensions to the original DQN framework: double Q-learning, dueling network architecture, multi-step bootstrapping, prioritized experience replay, distributional Q-learning, and noisy networks for exploration. The Q-network is structured as a dueling architecture with separate value and advantage streams,

each composed of fully connected layers, culminating in a categorical distribution over discrete support atoms to model the value distribution. Exploration is handled through noisy linear layers, eliminating the need for ϵ-greedy exploration.

We compute Q-targets using multi-step returns, and the target network is updated using double Q-learning logic, where the online network selects the greedy action and the target network evaluates it. Transitions are sampled from a prioritized replay buffer with importance sampling corrections to reduce bias from non-uniform sampling. The distributional Bellman projection is used to compute the target distribution over atoms. During evaluation, we disable noise in the network to assess the agent's learned policy without stochasticity. This unified architecture is designed to improve learning stability, efficiency, and performance by combining complementary DQN advancements.

4 Experimental Setup

Experiment Overview Experiments are designed to increase in difficulty. The simplest environment has no walls, a fully dirty layout, and of grid size of (1xN). An intermediate-difficulty environment has no walls, a fully dirty layout, and of rectangular size such as (20x20). A more realistic environment is set to have walls, a fully dirty layout, and of rectangular size. We report two sets of environments for this scenario: one with a 20x20 grid size and four rooms with random door locations, and another with a 40x30 layout with custom furniture and wall setups. The most challenging environment is set to have walls, but with only several dirt clusters, and of rectangular size. We report one experiment result on a 20x20 grid size with four rooms and five clusters of dirt. We designed different evaluation layouts that mimic the training layout, such as how the walls are constructed, for each scenario.

Sanity Checks Before starting the experiments, we ran our algorithms on the simplest environment of size (1xN) as a sanity check and algorithmic verification. All three algorithms passed the sanity check that they cleaned up all the grids and terminated. Illustrations in the Appendix in Figure 6 display the sanity check environment and the last rendered step before the episode terminates.

Training and Evaluation Configurations In the following more difficult experiments, we set a maximum time limit of 3000 time steps for each episode so that the current trajectory would end if the agent is stuck for a prolonged time. To prevent wasting the replay buffer for undesired behaviors in the beginning of training, we also set a truncation rule if the agent stays at the same tile consecutively for more than 128 time steps. Both of these configurations facilitate learning from meaningful experiences. During the evaluation phase, we also allow 3000 maximum time steps before ending one evaluation episode and calculating evaluation metrics. All evaluations are conducted on 5 episodes. The total training time for each algorithm and environment ranges from 500,000 time steps to 2,500,000 time steps, or until convergence.

Hyperparameter Tuning To facilitate more efficient training, we constructed hyperparameter tuning pipelines with `Optuna` that automatically search the hyperparameter space using a Bayesian optimization sampling method that is more efficient than grid search. Specifically, we tuned learning rate and discount factor for all algorithms; number of time steps before updating policy, entropy coefficient bonus used in exploration, and clip range for PPO; buffer size, batch size, exploration decay fraction, and the starting and end exploration probability for DQN. For Rainbow DQN specifically, we additionally tuned several critical hyperparameters through `Optuna`, including the number of atoms for distributional RL, the value distribution support bounds, and the number of steps used in multi-step returns. We also optimized the learning rate (lr), discount factor (gamma), batch size, and prioritized experience replay settings. Furthermore, we experimented with different environment wrappers via the categorical parameter `wrapper_type`, allowing us to assess the impact of observation preprocessing and reward shaping on agent performance.

Evaluation Metrics The following evaluation metrics are used to evaluate model performance:

- **Coverage Ratio (CR)**: the proportion of dirty grids that are cleaned during the operation. It is defined as:

$$\text{CR} = \frac{N_{\text{cleaned}}}{N_{\text{dirty}}} \tag{5}$$

  where N_{cleaned} is the number of unique dirty grids cleaned by the agent, and N_{dirty} is the total number of dirty grids in the environment.

- **Redundancy Ratio (RR)**: the proportion of redundant visits to grids over the total number of steps. It is defined as:

$$\text{RR} = \frac{N_{\text{steps}} - N_{\text{visited grids}}}{N_{\text{steps}}} \tag{6}$$

  where N_{steps} is the number of timesteps in this episode, and $N_{\text{visited grids}}$ is the number of visited grids in the environment.

- **Revisit Ratio (RV)**: the average number of visits to grids.

$$\text{RV} = \frac{N_{\text{steps}}}{N_{\text{grids}}} \tag{7}$$

where N_{steps} is the number of timesteps in this episode, and N_{grid} is the total number of unique grids in the environment.

Together, these metrics provide a more holistic view of the learned cleaning policy that accounts for both coverage and efficiency compared to reward alone. With multiple episodes of evaluations, we report both the mean and standard deviation of these metrics.

Baseline We implemented an agent that takes a random action every step as the baseline. The agent is evaluated in a similar way, with a maximum of 3000 time steps per episode over 5 episodes.

5 Results

5.1 Quantitative Evaluation

Environment	Method	Best CR	CR	RR	RV
20x20	Random	0.80	0.67 (0.12)	6.36 (1.68)	17.80 (14.00)
	PPO	1.0	0.99 (0.02)	2.82 (0.49)	0.64 (0.07)
	DQN	0.95	0.83 (0.02)	0.82 (0.10)	4.66 (2.40)
	Rainbow	0.80	0.38 (0.32)	2.74 (2.85)	169.09 (180.56)
20x20 Walls	Random	0.69	0.64 (0.05)	7.99 (0.18)	14.96 (1.79)
	PPO	0.95	0.88 (0.12)	2.12 (0.55)	0.57 (0.08)
	DQN	0.58	0.46 (0.13)	3.68 (1.67)	60.15 (50.82)
	Rainbow	0.46	0.29 (0.16)	3.49 (2.86)	109.93 (92.13)
40x30 Custom Walls	Random	0.48	0.39 (0.12)	2.37 (0.51)	80.98 (43.57)
	PPO	0.82	0.74 (0.08)	1.28 (0.15)	0.42 (0.03)
	DQN	0.56	0.42 (0.14)	1.26 (0.57)	191.98 (160.14)
	Rainbow	0.25	0.10 (0.09)	0.92 (1.08)	906.46 (885.85)
20x20 Dirt Clusters	Random	0.67	0.49 (0.17)	7.22 (1.10)	17.19 (4.07)
	PPO	0.68	0.53 (0.14)	1.44 (0.34)	0.63 (0.04)
	DQN	0.73	0.24 (0.26)	1.63 (1.17)	222.06 (148.10)
	Rainbow	0.70	0.42 (0.19)	4.45 (2.04)	65.21 (57.33)

Table 2: Performance evaluation across environments. Best run coverage ratio (Best CR) and average with standard deviations (in parentheses) are reported for each metric.

The performance results (in Table 2) across diverse environments highlight notable trends among the evaluated methods. PPO consistently achieves the highest coverage ratios, particularly in structured settings such as the 20×20 grid with or without walls, where it achieves near-perfect mean coverage (0.99 and 0.95, respectively). This aligns with PPO's robustness in dense reward settings and its ability to handle continuous updates via clipped policy optimization. However, PPO also exhibits extreme variance in reward, especially under wall constraints and sparse rewards, likely due to sensitivity to poor reward shaping and the lack of explicit value normalization.

Rainbow DQN, while underperforming PPO in raw coverage, exhibits some unique strengths under specific structural priors. In the 20×20 clustered dirt environment, Rainbow DQN outperforms DQN and Random in coverage and efficiency, suggesting that its integration of n-step returns, distributional Q-learning, and prioritized replay allows for more efficient credit assignment and learning from informative transitions. However, in larger or more complex environments like the 40×30 custom wall setup, Rainbow's performance drops significantly. The revisit ratio spikes and coverage drops to 0.10 on average, suggesting that its exploration strategies (e.g., noisy networks) may be insufficient in complex mazes without structured reward guidance.

Standard DQN performs more stably than Rainbow in large-scale settings (e.g., 40×30), potentially due to simpler training dynamics. This could be attributed to better reward-tuning due to a simpler search space of hyperparameters. Random policies, although generally poor in efficiency, surprisingly reach moderate coverage in some cases (e.g., 20×20 no-wall).

PPO's effectiveness highlights the value of policy-gradient methods in structured, reward-rich environments, especially when shaped rewards align well with multi-phase objectives. Meanwhile, Rainbow and DQN illustrate the trade-offs between architectural complexity and stability, with DQN sometimes benefiting from its simplicity and robustness in environments where Rainbow's exploration and learning enhancements fail to activate effectively.

5.2 Qualitative Analysis

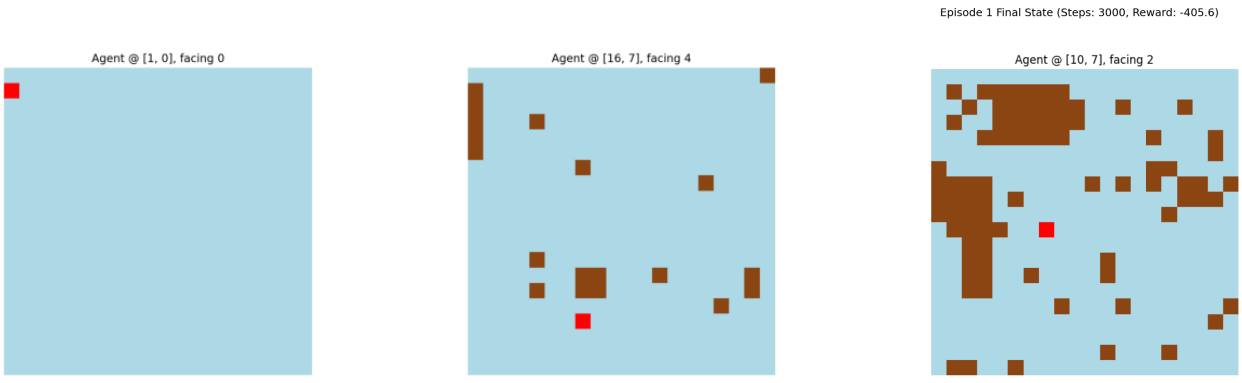
We present sample of terminating frame for each algorithm in each environment in Figure 1, Figure 2, Figure 3, Figure 4.

Across all environments, PPO demonstrates the most structured and goal-consistent behavior, often terminating in positions suggesting complete or near-complete coverage. Its ability to learn reliable navigation policies is particularly evident in cluttered or large-scale environments, where it avoids repeated states and navigates around obstacles with minimal redundancy. This behavior aligns with PPO's strength in trajectory-level optimization and effective exploitation of shaped rewards, enabling the agent to generalize its exploration across environment geometries. Moreover, after it earns a reward for completing the cleaning or returning home, it efficiently learn to take fewer timesteps to achieve such a goal. In the 1x40 grid environment, the total timestep converges to 82, the optimal timesteps to clean and return. As the environment gets more complex, the time to final reward and termination gets longer, but once the termination goal is achieved, PPO can optimize to use fewer steps.

For DQN, when in the open layouts, such as the 20×20 no-wall scenario, the agent demonstrated reasonable coverage behavior, suggesting that it had learned general exploration strategies that allow it to sweep through the whole space. DQN also helped the agent to learn to navigate with walls to some extent, as evident in the 40×30 with custom wall environment. However, in more constrained (e.g., narrow passage) or structured environments (e.g., complex maps), such as the 20×20 with walls or the 20×20 with dirt cluster scenarios, DQN frequently terminated with visible dirty tiles nearby or remained confined to one part of the whole space. This indicates difficulties in leveraging the dirt map in its observation to guide goal-directed cleaning and highlights limitations in long-term planning. Despite incorporating reward shaping and exploration bonuses, the agent's policy remained sensitive to layouts. Some evaluation trajectories indicated the agent spent meaningless steps turning to different orientations and got stuck around corners, surrounded by walls. The error analysis suggests the need for richer observation processing, improved exploration mechanisms, or architectural enhancements, all with spatial information incorporated (e.g., convolutional or recurrent layers).

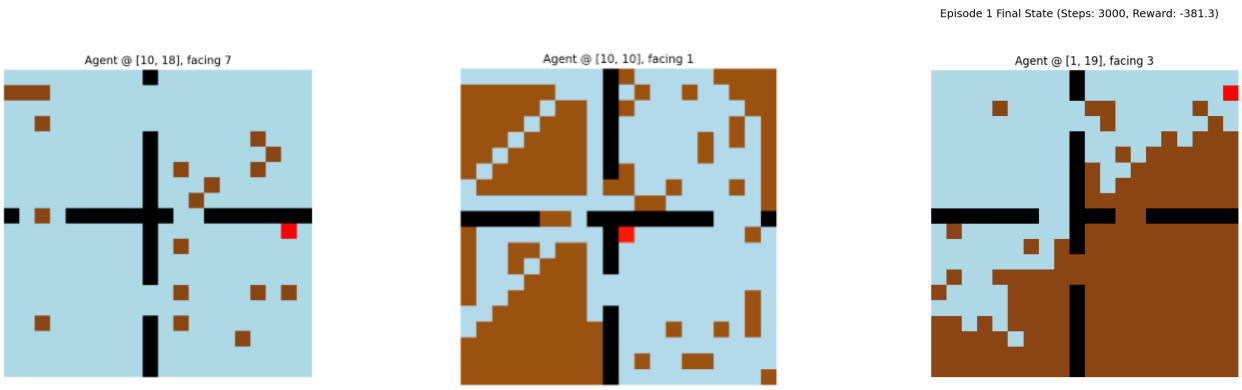
Rainbow DQN shows highly variable performance. In simpler environments, Rainbow sometimes explores more broadly than DQN, helped by the prioritized replay and multi-step returns. However, its behaviour often lacks spatial coherence. In cluttered or large environments, Rainbow tends to produce erratic trajectories with high revisit rates (as seen in Table 2), suggesting that its exploration mechanisms (e.g. noisy nets) and complex value targets fail to scale without stronger structural priors or dynamic reward adjustments.

Observing the evaluation runs, PPO performs better than DQN and Rainbow DQN in some environments, while all of them have similar performance in the most challenging case with dirt clusters. PPO is set to be different from DQN and Rainbow DQN because it is an on-policy algorithm, learning from collected data for the current policy training. Based on the obstacle setups, the best policy is sensitive to recent data, and reusing data from previous episodes might be less efficient. In the more difficult case where there exists walls separating different rooms, the door position significantly impacts the best strategy.



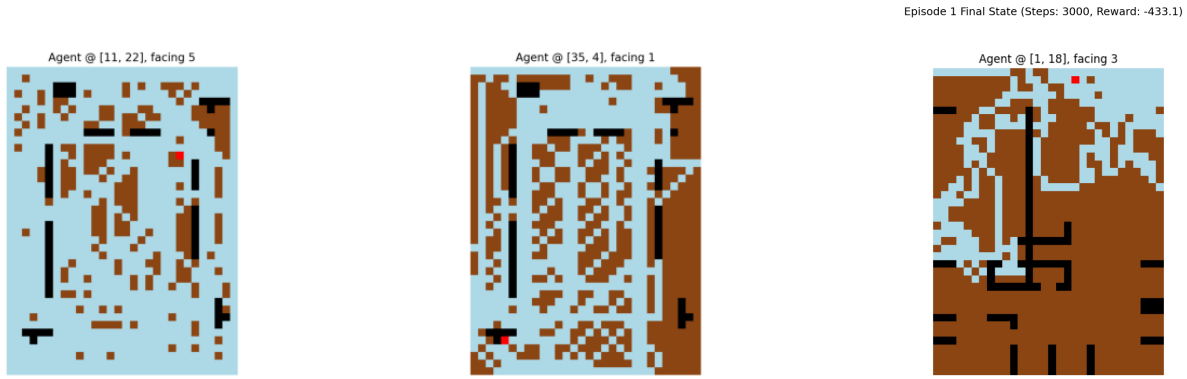
(a) PPO (b) DQN (c) Rainbow DQN

Figure 1: Terminating frames of 20x20 no wall environment.



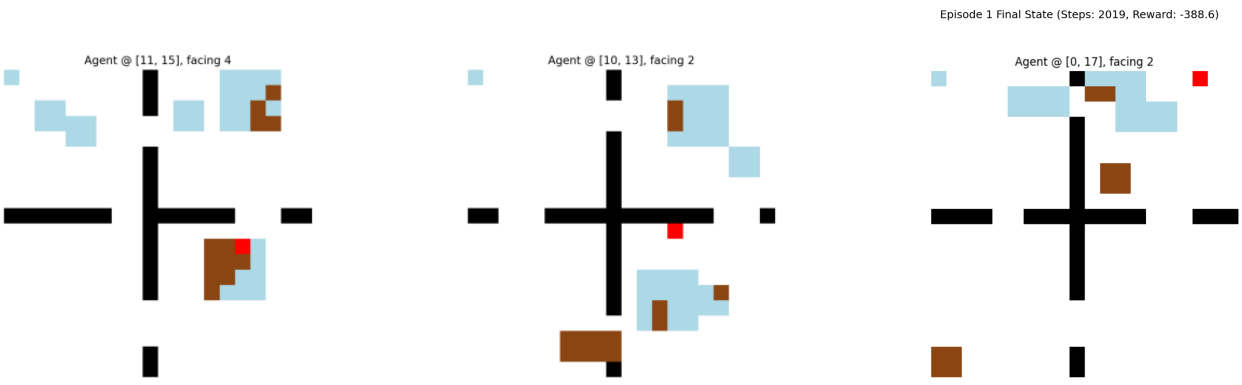
(a) PPO (b) DQN (c) Rainbow DQN

Figure 2: Terminating frames of 20x20 with wall environment.



(a) PPO (b) DQN (c) Rainbow DQN

Figure 3: Terminating frames of 40x30 with custom wall environment.



(a) PPO (b) DQN (c) Rainbow DQN

Figure 4: Terminating frames of 20x20 with dirt map environment.

6 Discussion

6.1 Reward Shaping and Behavioral Trade-Offs

One of the key challenges during the experiments was reward tuning. Due to the multi-phase nature of the task, namely first maximizing coverage through efficient exploration, then returning home when cleaning is complete, the agent's behavior must be guided not just toward goal achievement, but also away from inefficient or non-terminating actions. In practice, this requires a carefully shaped reward function that incorporates both positive incentives for desired behaviors, such as discovering new tiles or dirty areas, and penalties for unproductive or costly ones, such as standing still, revisiting cleaned tiles.

We found that PPO models responded well to heavily penalized rewards, especially with the inclusion of the exploitation penalty wrapper that discouraged stagnation and helped the agent develop structured trajectories. In contrast, DQN models benefited more from exploration incentives, such as visit-based and exploration-based bonuses that promoted broader spatial coverage early in training.

More specifically, the multi-phase nature of the task can be framed into two phases: (1) efficient coverage and (2) efficient return to the starting position. For agents capable of completing cleaning tasks, a distance-based reward shaping component that rewards shorter paths back to the home base may better encourage the return behavior than relying solely on learning the penalization of stagnation. In our ablation analysis, we observed that PPO agents were only able to return home when both exploration bonus and exploitation penalty shaping wrappers were active. Once the exploitation penalty wrapper was removed, PPO failed to return to base, suggesting that surrogate mechanisms like this are fragile. A more direct and explicit reward design may be required for reliable completion of both phases.

For Rainbow DQN, we experimented with two reward shaping wrappers, the simple and the smart, each modifying the environment's reward structure in distinct ways. The simple wrapper applies static shaping rules: it penalizes the agent for no-op actions (i.e., standing still), revisiting cleaned tiles, and rewards visiting new or dirty tiles. These rules are fixed and do not adapt based on the agent's progress through the task. In contrast, the smart wrapper incorporates phase-aware logic that dynamically alters reward priorities based on task progression. It tracks the agent's cleaning progress, measured as the percentage of the environment cleaned, and switches reward focus once a predefined threshold is crossed. During the initial exploration phase, the agent receives rewards for visiting dirty or unexplored tiles. Once sufficient coverage is achieved, the wrapper transitions to a return phase, introducing a distance-based penalty (e.g., Manhattan distance to the home base) that incentivizes the agent to return efficiently. This structured, two-phase reward logic allows the smart wrapper to provide clearer learning signals for tasks with sequential objectives.

From hyperparameter tuning and evaluation, we found that Rainbow DQN agents trained with the smart wrapper consistently outperformed those using the simple wrapper. This is likely due to Rainbow's multi-step bootstrapping and distributional Q-learning, which propagate reward signals more effectively over time. The phase-aware structure of the smart wrapper produces clearer reward gradients aligned with task stages, allowing Rainbow to learn more structured and complete behaviors that satisfy both exploration and return objectives.

These findings highlight the importance of phase-aware reward shaping, especially in tasks with multi-objective goals like this one. Future work may benefit from hierarchical or adaptive reward structures or wrappers that shift focus dynamically between different goals.

6.2 Memory and Observations

We started the experiments with the agent only knowing local observations of the eight nearby grids. Training is done through accumulated rewards as the agent moves. The fact that it lacks memory of grids it explored and that it has no knowledge of what grids need to be covered produces significant challenges for all algorithms to perform better than random.

We then include extra information in the observation space: a map that the agent uses to keep track of observed obstacles during transitions, and a dirt map that specifies grids that need to be cleaned. This is beneficial in aiding the agent to learn the relationships between the goal and its position and has been proven to improve model performance in the later experiments.

6.3 Variable Encoding and Convolution

Since we work with a spatial environment, encoding spatial information in a useful format is crucial for learning. We experimented with different types of encoding and concluded that two major encoding decisions, including the environment layout and the agent status, are crucial for model performance.

The layout of a 2D map is encoded as a 2D array and then flattened into a vector for normalization. The flattening process destroys the spatial structure of the environment, potentially causing problems for learning. Convolutional layer helps preserve spatial information and transforms a multi-channel multi-dimensional vector into a concatenated one-dimensional vector.

To encode the agent position, a one-hot encoding of the agent position and start position is used and then passed through a convolutional layer. Compared with a (x, y) encoding, this encoding preserves the spatial proximity for nearby grids. To encode

the agent orientation, a 2D vector of sine and cosine is used. Compared with dictionary encoding, the radius measurement preserves spatial information and is continuous.

While a convolutional layer may improve overall performance, it significantly increases the computation needed and limits the number of steps each algorithm can be trained or tuned for. A 10x10 grid environment with walls is used to compare performance using the new encoding scheme. The qualitative result is included in Figure 5.

6.4 Training and Tuning Process

We found that during the experimentations, while automated hyperparameter tuning using `Optuna` offered a systematic approach to improving training efficiency, the overall impact on learning is often limited and uneven. This aligns with previous studies on DRL's sensitivity to stochastic factors and initializations frequently led to inconsistent gains across trials (Henderson et al., 2019; Islam et al., 2017). In response, we added multiple seeds per sampled configuration to reduce variance and better estimate stability, though this substantially increased compute cost, limiting our ability to perform a finer-grained tuning after an initial sweep of the hyperparameter space.

To better align tuning with downstream desired performance, we also experimented with evaluating each configuration using a weighted score that combines coverage ratio and total episode reward. This composite objective encouraged balanced policies and provided more interpretable tuning criteria. However, selecting the right trade-off weights remained a manual process that required tuning. The optimal balance was also sensitive to environment layouts and episode lengths. While this approach had limited benefit in our study, we believe future work involving more complex objectives should further explore this direction.

We also observed that some of the best-performing configurations during tuning were locally optimal and produced high returns or coverage at the fixed tuning horizon but failed to generalize beyond that. This temporal locality meant the tuning process could be biased toward short-term exploitation behavior and potentially overlook parameter sets that would yield stronger long-term learning. While this is a known limitation in early-stopping based RL tuning (Li et al., 2018), it is particularly significant in value-based methods like DQN, where training dynamics can shift sharply over time. Overall, tuning improved reproducibility and structure in our training pipeline, but due to the large search spaces and high variance of DRL, its marginal returns diminished quickly.

7 Conclusion

Our study demonstrates that reinforcement learning can be effectively applied to coverage path planning for cleaning robots, especially when algorithms are paired with well-shaped, task-aware rewards. Among the models evaluated, PPO consistently performed best, achieving near-perfect coverage in structured environments and exhibiting reliable return-to-base behavior. This emphasizes the advantage of on-policy methods in dynamic environments, where recent experiences more accurately reflect the task context.

Reward shaping emerged as a critical factor, especially in guiding multi-phase tasks. Tailored wrappers enabled the agents to balance exploration with termination objectives. Rainbow DQN, despite its theoretical strengths, showed high variance and instability in complex environments, emphasizing the importance of aligning architecture complexity with reward signal clarity and environment structure. DQN, while simpler, offered more stable but limited generalization.

Additional contributions include memory-enhanced observations and convolutional encodings that improved spatial reasoning, though they introduced significant computational overhead. Despite automated hyperparameter tuning, reinforcement learning remains highly sensitive to initialization and environmental variability, which limits tuning effectiveness in practice.

Future directions include more expressive memory architectures and exploration strategies beyond reward shaping, such as curriculum learning, to enable more scalable and transferable coverage policies. Another promising avenue is the systematic integration of convolutional neural networks (CNNs) with value-based methods like DQN and Rainbow, which have shown strong compatibility in spatial domains. CNNs can enhance state representations by capturing local spatial patterns, enabling more efficient learning and better generalization in complex or partially observable environments. Combining these approaches with robust reward design may significantly improve the stability and effectiveness.

8 Team Contributions

- **Ngoc Vo** : designed and implemented Rainbow DQN
- **Yiran Fan** : implemented DQN and environment
- **Siqi Ma** : implemented PPO and environment

Changes from Proposal Our focus changed from implementing a variety of environment layouts (e.g., irregular grids) to algorithmic improvements, such as reward shaping, designing additional components to common DRL methods, and conducting experiments with different difficulty levels.

References

José Pedro Carvalho and A. Pedro Aguiar. 2025. Deep reinforcement learning for zero-shot coverage path planning with mobile robots. *IEEE/CAA Journal of Automatica Sinica* (2025), 1–16. https://doi.org/10.1109/JAS.2024.125064

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2019. Deep Reinforcement Learning that Matters. arXiv:1709.06560 [cs.LG] https://arxiv.org/abs/1709.06560

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2017. Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv:1710.02298 [cs.AI] https://arxiv.org/abs/1710.02298

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. 2017. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. arXiv:1708.04133 [cs.LG] https://arxiv.org/abs/1708.04133

Arvi Jonnarth, Jie Zhao, and Michael Felsberg. 2024. Learning Coverage Paths in Unknown Environments with Deep Reinforcement Learning. arXiv:2306.16978 [cs.RO] https://arxiv.org/abs/2306.16978

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. arXiv:1603.06560 [cs.LG] https://arxiv.org/abs/1603.06560

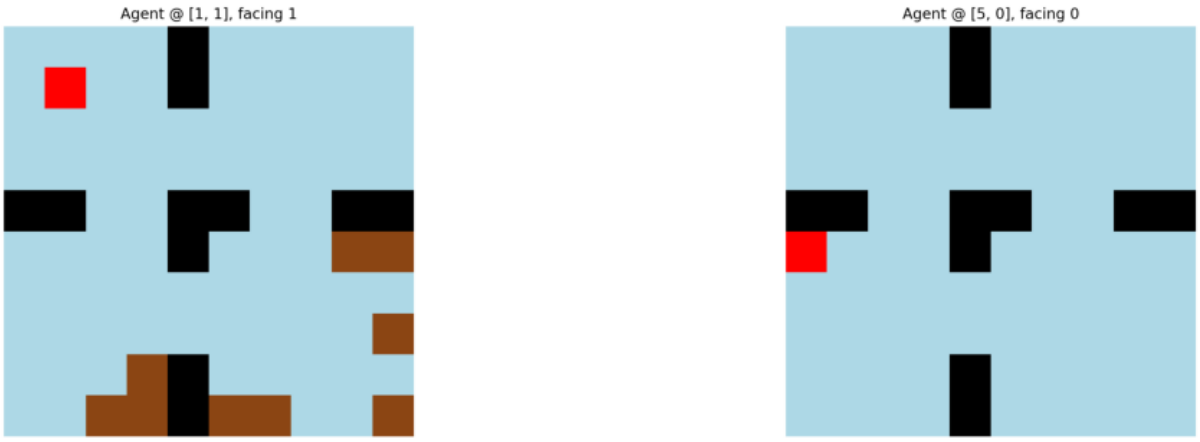
Woohyeon Moon, Taeyoung Kim, Bumgeun Park, Dongsoo Har, and Sarvar Hussain Nengroo. 2022. Path Planning of Cleaning Robot with Reinforcement Learning. *arXiv preprint arXiv:2208.08211* (2022).

Hao Qin, Bing Qiao, Wenjia Wu, and Yuxuan Deng. 2022. A Path Planning Algorithm Based on Deep Reinforcement Learning for Mobile Robots in Unknown Environment. *IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (2022).

Miguel Quiñones-Ramírez, Jorge Ríos-Martínez, and Víctor Uc-Cetina. 2023. Robot path planning using deep reinforcement learning. *arXiv preprint arXiv:2302.09120* (2023).

A Additional Experiment

To evaluate the usefulness of encoding observation into multidimensional arrays and perform feature extraction through convolutional layers, an experiment is run on a 10x10 simple grid with walls using PPO for quick verification.



(a) CNN Feature Extractor (b) Flattened Array

Figure 5: 10x10 grid experiment with CNN or flattened array.

B Implementation Details

B.1 Code Availability

The code used in this project can be found at this GitHub link: https://github.com/ma-siqi/224r_project.

B.2 Sanity Check Environment

(a) Environment (b) Terminating Frame

Figure 6: An illustration of 1x40 sanity check environment.

B.3 PPO Reward Shaping Wrapper

Reward	Type	Reward Design
-0.5	Undesired	Revisting a grid more than once
-0.1	Undesired	Stay in place

Table 3: Reward wrapper for PPO.

B.4 DQN Reward Shaping Wrapper

Reward	Type	Reward Design
$+0.3$	Desired	Visit a new grid
$+0.1$	Desired	Observe a new grid
$+0.3$	Desired	Enter a tile that is adjacent to unexplored tiles
$+0.3$	Desired	Move away from obstacle
$+0.1$	Desired	Re-orientation to face a dirty grid
$+0.1$	Desired	Re-orientation to face an unknown grid
-0.5	Undesired	Spinning
-0.05	Undesired	Stay in place
-0.001	Undesired	Face obstacles or invalid next move directions

Table 4: Reward wrapper for DQN.

B.5 Rainbow DQN Reward Shaping Wrapper

Reward	Type	Reward Design
		Phase 1: Exploration
$+0.3$	Desired	Visit a new grid
$+0.1$	Desired	Observe a new grid
$+0.3$	Desired	Visit frontier tile
$+0.3$	Desired	Move away from obstacle
$+0.1$	Desired	Re-orient to face a dirty grid
$+0.1$	Desired	Re-orient to face an unknown grid
		Phase 2: Return-to-Base
$-0.01 \times d$	Undesired	Distance penalty to home base (Manhattan distance d)
		Universal Penalties (Both Phases)
-0.5	Undesired	Spinning in place
-0.05	Undesired	Staying idle
-0.001	Undesired	Facing obstacles or invalid moves

Table 5: Reward wrapper for Rainbow DQN using the Smart Wrapper. Phase-aware logic transitions to a return policy once coverage exceeds a set threshold.