Universiteit Leiden

Master Computer Science

A Benchmark Study of Deep Reinforcement Learning Algorithms for Container Stowage Planning Problem

Name:	Yunqi Huang
Student ID:	s3789918
Date:	27/08/2025
Specialisation:	Data Science
1st supervisor:	Yingjie Fan
2nd supervisor:	Aske Plaat

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

ABSTRACT

Container stowage planning is an important component of maritime transportation and terminal operations, directly impacting the efficiency of the supply chain. Because the problem is complex, it heavily relies on the expertise of human planners. With the rise of artificial intelligence, some studies have attempted to address container stowage planning problems (CSPP) using reinforcement learning (RL) methods. However, there is still a lack of benchmark comparisons across different RL approaches. To fill this research gap regarding the performance differences of DRL methods on CSPP, this study develops a Gym environment that abstracts the fundamental features of CSPP. By incorporating crane scheduling, the study further extends this into two formulations, multi-agent and single agent, to jointly optimize CSPP and crane scheduling. Based on this setup, the study designs multiple scenarios to conduct comparative experiments on five RL algorithms: DQN, QR-DQN, A2C, PPO, and TRPO. The experimental results show that all five algorithms perform well in optimizing key metrics in simple environments, while TRPO stands out in highly complex scenarios. Moreover, in complex settings, the single-agent control method performs better in reducing shifters. Overall, this study demonstrates how the performance of RL algorithms in solving CSPP is nuanced by scenario complexity and problem formulation, emphasizing the importance of choosing appropriate approaches according to the specific characteristics and requirements of the problem in practical applications.

ACKNOWLEDGEMENTS

This thesis was produced in collaboration with Loadmaster.ai and supervised by the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University, made possible using the computational resources of the Academic Leiden Interdisciplinary Cluster Environment (ALICE).

I would like to thank my supervisors, Yingjie Fan and Aske Plaat, for their guidance, support, and encouragement throughout the writing of this thesis. I truly appreciate our meetings and email exchanges, which have been very inspiring and gave me a lot of confidence.

I am also grateful to my colleagues at Loadmaster.ai for their help throughout the project, from whom I learned a great deal both in research and engineering. Special thanks go to Alexis Carras for providing various ideas, Nishith Chennakeshava for guiding me to think more deeply about the problems, and Vladislav Neverov for his valuable feedback.

Finally, I would like to thank my friend Shuo Chen for the care and support during this time.

Completing this thesis would not have been possible without all of you, and I am sincerely grateful.

Contents

1 Introduction

Container ports are hubs connecting maritime and land transport and serve as an integral component of the global supply chain. The allocation, coordination, and loading of containers directly impacts the efficiency of the terminals within the broader logistics network (He et al., 2023). In recent years, container terminals have been faced with increasing demand for throughput and transportation volumes, resulting in the need for faster container handling and turnaround times (Zhou et al., 2022). Vessel loading operations is particularly important for the efficient terminal management, which involves an critical challenge: the container stowage planning problem (CSPP), which is, to find the optimal approach for loading containers from the yard to the vessel under a given set of constraints. CSPP is an NP-hard problem (Avriel and Penn, 1993), subject to various constraints including vessel structural characteristics, container dimensions, stowage regulations, customer-specified requirements for containers, and seaworthiness considerations.

In container terminal operations, stowage planning represents a critical and time-intensive phase, and it relies heavily on the expertise of human planners (Jensen et al., 2018). Manual planning has become a bottleneck for terminal automation, especially as vessel and yard capacities continue to increase, leading to higher labor and time consumption (Shen et al., 2017). The planning challenges call for smarter solutions: from Review of Maritime Transport 2024 by UN Trade and Development (UNCTAD), the number of containership port calls has reached a historical high, while ports adopting AI and automation technologies reported less waiting time and faster cargo turnover (Trade and Development, 2024).

Early research on CSPP solutions focused on the exact methods using mathematical models. For example, Li et al. (2008) using a 0-1 linear programming model and branch-and-cut method to maximize vessel load while minimizing container shiftings. However, no exact method or mathematical model has yet proven efficient enough to solve real-world single-port CSPP large instances (van Twiller et al., 2024). For more complex constraints and larger problem sizes, some research shift to metaheuristics methods, such as neighborhood heuristics (Ambrosino et al., 2010) and population heuristics (Hu et al., 2012).

Heuristic methods often build mixed integer programming models for specific cases and designing algorithms to solve them accordingly, which may limit the performance over large-scale instances and their generalization ability (Shen et al., 2017) . Recent studies have begun to explore reinforcement learning (RL) approaches for CSPP (Jiang et al., 2021; Wei et al., 2021), but most focus on the performance of a single algorithm in a specific environment. Due to data scarcity in the CSPP domain (Jensen et al., 2018), there is a lack of benchmark comparisons across different algorithms solving the same optimization objective in a shared environment.

Another aspect to consider in CSPP research is joint optimization, such as including equipment scheduling. Since terminal operations are tightly coupled across multiple stages, focusing on a single stage may yield limited improvements or even negative effects for overall optimization (Hsu et al., 2021). Given that crane operation on the quay side and container handling on the yard side are both part of the container loading process, joint optimization of CSPP and crane scheduling constitutes an important aspect of whole terminal operation. However, research in this area is relatively scarce (Kizilay and Eliiyi, 2021), and there is no exploratory work using reinforcement learning to solve this problem.

In scenarios where multiple cranes load a single ship in parallel, the joint control problem of

CSPP and crane scheduling can be modeled as either a single-agent central control problem, where one agent selects both containers and cranes, or a multi-agent problem where each agent representing a crane makes container selection decisions respectively. The impact of these two formulations on the performance of different RL algorithms remains to be studied.

To address the above challenges, this study explores the core question: Can reinforcement learning be applied to solve the CSPP? On this basis, we break it down into the following sub-questions:

1. *How do different RL algorithms perform regarding the container stowage planning problem in the same environment?*
2. *How does problem scale affect the performance of RL algorithms?*
3. *When crane scheduling is considered together with CSPP, how do single-agent vs. multi-agent formulations influence algorithm performance?*

This thesis will elaborate on the complex aspects of the stowage planning problem in Section 2, including domain knowledge related to container ships, loading procedures, and problem constraints. Section 3 provides an overview of existing work on solving CSPP, covering both traditional methods and reinforcement learning approaches, as well as studies on the joint optimization of CSPP and crane scheduling. Section 4 introduces the preliminaries of reinforcement learning. Section 5 presents the research methodology, including the construction of the benchmarking environment, the algorithms compared, the use of action masks, also the experimental setup. Section 6 presents the experimental results, and Section 7 discusses the findings. Section 8 concludes the paper and outlines directions for future research.

2 Container Ship Stowage Planning Problem

This section provides a brief introduction to the structure of container ships and problem constraints. For two types of container ships, Ro-Ro (Roll on-Roll off, which uses ramps or ferry slips at the bow or stern for cargo handling) and Lo-Lo (lift on-lift off, which uses cranes for loading and unloading) (Garratt, 1980), the optimization problem in a single-port scenario for Lo-Lo will be focused on.

2.1 Structure of Container Ship

Consider a typical containership layout as shown in Figure 1. The smallest unit for loading containers to a ship is a slot, with common dimensions of 20 feet long, 8 feet wide, and 8 feet high. The coordinates of a single slot on a container ship are given by a triplet (bay, row, tier). The bay coordinate refers to the horizontal location in relation to the ship's side, usually counted from the bow to the stern. The row coordinate represents the horizontal position in relation to the front of the ship, counted from the center to the outer edge of the ship. The tier coordinate represents the vertical position relative to the ship's body, typically counted from bottom to top. Additionally, the tiers of vessel slots can be divided into two main sections: the portion located below the deck within the ship's hull, and the portion located above the deck. They are separated by hatches, and whether the hatch is open or closed determines the accessibility of the tier levels above it.

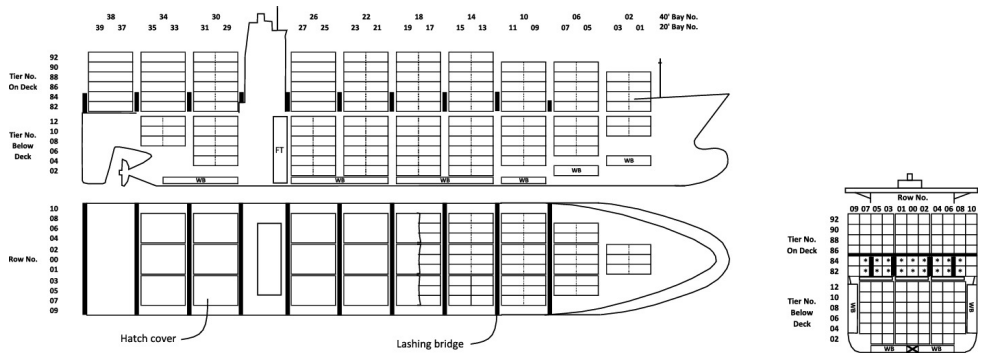


Figure 1: *Vessel structure* (van Twiller et al., 2024).

2.2 Stowage Operation Process

The operation process of container stowage depends on the terminal layout, but can generally be abstracted as shown in Figure 2. The key nodes are:

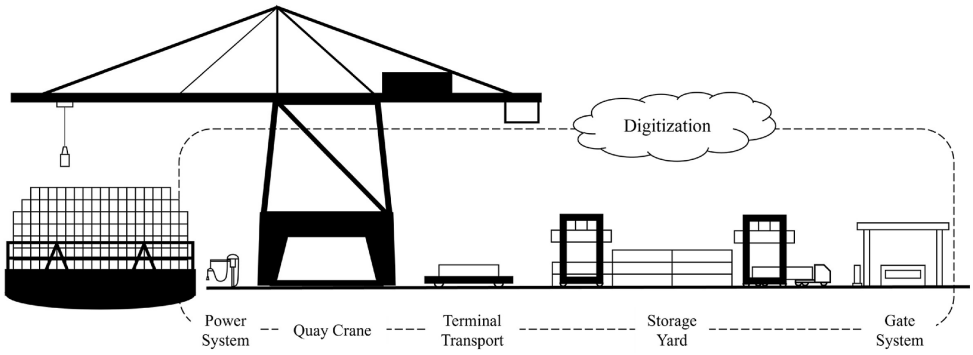


Figure 2: *Stowage operation process* (Zhou et al., 2022).

1. Storage Yard: This is where containers awaiting loading are stored, having a coordinate system similar to container vessels, with the minimum unit being a slot of 20 feet length, 8 feet width, and 8 feet height. The coordinates of the slot are represented by a triplet of bays, rows and tiers. The yard is typically divided into different areas, which is usually reflected in the yard slot indices. After the completion of pre-stowage, containers in the yard are typically placed in an orderly manner according to category, weight, and whether they are designated for the upcoming vessel to be loaded.

2. Handling Equipment: Used for lifting containers and moving them between different locations within the terminal. These movements include not only directly transporting containers from yard to quay crane, but also repositioning containers within the yard, which often occurs when the container to be loaded is not at the top of its yard stack and cannot be directly accessed by the handling equipment. The operation of moving other containers within the yard to access the currently needed container is called a shifter or reshuffling.

   Common handling equipment includes manually operated straddle carriers (SC), container trucks (CT), as well as automated lifting vehicles (ALV), intelligent guided vehicles (IGV) and autonomous trucks (AT) that don't require manual operation. (Zhou et al., 2022)

3. Quay Crane: Used for loading containers from the quay to the vessel at the seaside. For a ship, there can be a single crane or multiple cranes performing loading operations at the same time.

4. Vessel: The destination of an individual container within the stowage operation process.

For a single container awaiting transport, it needs to go through several steps: shifting, pickup and transport to the quay crane area by handling equipment, waiting for crane availability, and loading onto the ship by the crane.

2.3 Problem Model

The CSPP is about determining how to place m containers belonging to set C into n available ship slots belonging to set S (Ambrosino et al., 2004). We can iterate through S using a sequencer, and for each element $s \in S$, select the most suitable container $c \in C$, which turns this into a sequential decision problem. That is, we need to find an ordered sequence of selected containers ($\mathbf{P} = \langle p_1, p_2, \ldots, p_m \rangle$) such that our objective of interest is maximized. Here, $p_i \in C$ represents the i-th container picked from the set of available containers C to be placed, corresponding to a discrete time step in the loading process, while $p_i \neq p_j$ for any two different positions $i \neq j$ in the sequence $\mathbf{P}$.

2.3.1 Problem Constraints

There are a number of constraints that need to be considered in the stowage planning process:

1. Slot and container availability constraint: When the sequencer iterates through the vessel slots, we need to ensure that the slot to be operated on is vacant and designated for loading. Similarily, for container selection from the yard, we need to ensure that the container exists in the yard and is included in the loading list.

2. Bay adjacency constraint: When loading containers into a designated slot, it is necessary to check whether the adjacent bay of this slot is occupied. if so, then this slot is invalid. It's because the vessel bays have such divisions: odd-numbered bays are 20-foot slots, while even-numbered bays are 40-foot slots. For example, for two adjacent 20-foot bays starting from the bow, the forward one is numbered 01, the aft one is numbered 03, with these two combined as a 40-foot bay position numbered 02.

3. Hatch status constraint: When loading a container, the validity of the vessel slots depends on the situation of related hatch in the bay: if the hatch is open, containers should be placed in the cargo hold; if the hatch is closed, only slots above the deck are valid.

4. Sequence constraint: To ensure the stability and safety of the ship, loading should start from slots with smaller bay, row and tier values, and the slots on the seaside should be loaded before those on the land side.

5. Weight constraint: The combined weight of the containers loaded onto the vessel should not exceed the upper limit of the ship's weight limit Q, while the loading weight of containers for each bay must also not exceed the bay's individual weight limit. In container loading, lighter containers should not be located below heavier containers.

2.3.2 Objective of Optimization

The optimization objectives of CSPP vary according to the actual conditions of terminal operations, but commonly focused goals are reducing shifters and the stowage operation duration. In this study, we also focus on these two key performance indicators.

We denoted the stowage operation duration for a single container as t_{Job}. According to section 2.2, it can be concluded that t_{Job} consists of the following components:

$$t_{\text{Job}} = t_{\text{Shift}} + t_{\text{Move to Load}} \tag{1}$$

where t_{Shift} represents the time spent on shifting, which can be 0, while $t_{\text{Move to Load}}$ represents the time spent on transporting the container from the yard to quay crane, waiting for the crane to be available, and crane loading. t_{Shift} is directly related to n_{Shift}.

2.3.3 Crane Scheduling

Containers are loaded onto the vessel using quay cranes according to the stowage plan. When a vessel is handled by multiple cranes, the loading operations can be parallelized. However, badly-planned loading sequence might lead to underutilization of cranes in parallelization.

We consider using k cranes to execute the loading plan for m containers. Let the set

CR contain k crane elements, denoted as cr. Taking crane scheduling into account when solving the CSPP, we need to consider not only the selection of the containers but also which crane is assigned to perform the loading for each container. In this case, we seek to find a sequence ($\mathbf{P} = \langle (p_1, o_1), (p_2, o_2), \ldots, (p_i, o_i), \ldots, (p_m, o_m) \rangle$), where each $p_i \in C$ denotes the i-th container selected from C and each $o_i \in CR$ is the crane assigned to handle p_i at sequence position i. Here, $p_i \in C$ still adheres to the constraint $p_i \neq p_j$ for any two different positions $i \neq j$ in the sequence $\mathbf{P}$, while $o_i \in CR$ does not need to satisfy a distinctness constraint, but crane o_i must be available at time step (or sequence position) i.

3 Related Work

There has been considerable research on how to solve the stowage planning problem, and as artificial intelligence becomes more prevalent, using reinforcement learning schemes to solve this problem has been gaining attention from researchers during the past years. The following section begins with a overview for some solutions to the stowage planning problem, with a special focus on the application of reinforcement learning to this problem. Since the performance of agents in reinforcement learning depends on environment design, we will also discuss the impact of training environments on agents.

3.1 Study on Stowage Planning Problems

Container stowage planning problem is essentially an NP-hard combinatorial optimization problem. In terms of planning modeling, mainstream research divides it into two subproblems: the Master Bay Planning Problem (MBPP, which is about assigning container groups to general positions on the ship) and Slot Planning Problem (SPP, which is about assigning containers to slots on the ship after MBPP). After Wilson and Roach (2000) introduced this hierarchical decomposition method, Pacino et al. (2011) constructed models capable of balancing

ship stabilization with loading and unloading operations in their study based on this problem decomposition approach.

In addition, the stowage planning problem can be categorized into single-port and multi-port forms (van Twiller et al., 2024). In the multi-port problem, it is necessary to consider the re-stowage operations that may be caused by the unloading and loading of containers when the ship reaches subsequent ports along the transport route. Meanwhile, the single-port problem can be regarded as a subproblem of the multi-port problem, where the focus is on minimizing the number of shifters and operation time during container loading at the current port.

To address the single-port stowage planning problem, research mainly focuses on exact and metaheuristic methods. To minimize loading time, a heuristic approach was proposed by Sciomachen and Tanfani (2003) that builds on the connection between MBPP and the three-dimensional bin packing problem. This approach was successfully tested on a large scale, using more than 1400 container instances.

Ambrosino et al. (2004) provided a thorough description of the constraints involved in MBPP, and proposed an exact 0–1 linear programming formulation to address the problem. However, they found that this model was not applicable for large-scale real-world cases. Based on this, they proposed the use of heuristic preprocessing and prestowing procedures, as well as relaxing certain constraints for improvement.

Cruz-Reyes et al. (2015) addressed the MBPP by developing strong upper and lower bounds to define a feasible solution space. One of the lower bounds was obtained by relaxing the integer programming model of MBPP, while the other three bounds were designed heuristically. The lower bounds include LB1: Estimation of the minimal loading time, and LB2: LP relaxation (relaxing the integer constraint $x_{i,j} \in \{0, 1\}$ to continuous constraints). The upper bounds include UB1: Estimation of the maximal loading time and UB2: Estimation based on a loading procedure considering weight constraints. After testing on three groups of instances, they concluded that LB1 and UB2 are the best combination.

Zhu et al. (2020) proposed a simplified CSPP model and gradually introduced real-world factors into the model, solving it through integer programming. They considered four factors: whether the container is empty, the state of hatch covers, container size, and container type. These factors were incorporated separately to form four models. Performance was evaluated in each of these four scenarios, and it was found that the model integrating all factors achieved the highest solution quality, but the solution time increased as the problem became more complex.

Some studies have considered stowage planning with crane scheduling: Zheng et al. (2010) addressed the quay crane scheduling problem (QCSP) and yard crane scheduling problem (YCSP) incorporating vessel stowage planning (VSP) and yard storage planning (YSP) by proposing a heuristic method to schedule an automated container handling system using twin 40' cranes. Azevedo et al. (2018) employed a genetic algorithm along with simulation and representation by rules to solve the integrated problem of 3D Stowage Planning and the QCSP for container ship.

3.2 RL in Stowage Planning

Due to the sequential decision nature of CSPP, it is suitable to use reinforcement learning as solutions, with an increasing number of recent studies focusing on how to obtain effective

decision policies through RL approaches.

Shen et al. (2017) used Deep Q-Learning for CSPP, considered nine features including container weight in the environmental state modeling, and designed reward functions based on the generated stowage planning's feasibility, reshuffling, and yard crane movement frequency to train the network. Xia et al. (2020) introduced Prioritized Experience Replay (PER) when using DQN to solve the container Loading Sequencing Problem to address sparse reward issues in dynamic environments. Zhao et al. (2018) was the first to use Monte Carlo Tree Search (MCTS) to solve CSPP, utilizing roulette wheel to design the simulation for stowage to achieve loading objectives. van Twiller et al. (2023) used Proximal Policy Optimization (PPO) to solve the Master Bay Planning Problem (MBPP) subproblem within CSPP, designing reward functions that consider factors such as hatch overstowage, restowing, and center of gravity deviation to maximize vessel utilization and reduce hatch overloading. Cho and Ku (2024) adopted PPO with a two-phase approach to handle CSPP, similarly considering center of gravity deviation and bay weight in the reward function during the bay selection phase, while minimizing rehandling in the row and tier placement phase.

Current research mainly focuses on minimizing container shifting and ensuring stowage plan feasibility, but the duration of stowage operations has received limited attention. It is worth noting that considering the duration of the stowage operation as a factor in the reward function may render the agent's policy learning process dependent on the accuracy of the simulated environment's time prediction.

3.3 The Role of Environment Design in RL Methods

Due to the limitations of real-world data in terms of sample inefficiency and high acquisition cost, simulator-based training is widely used in the RL field to obtain an abundant source of data. However, the discrepancies between simulated and real environments often impact policy transfer performance (Zhao et al., 2020). This raises questions about how the training environment affects training effectiveness and generalization.

Henderson et al. (2018) discovered in their research on RL experiment reproducibility that environment characteristics can significantly affect RL algorithm performance. For instance, DDPG performs notably better in environments with stable dynamics compared to dynamically unstable environments (such as Hopper-v1). Padakandla (2021) pointed out that in non-stationary environments, RL agents encounter challenges in sample efficiency when the environment changes, and model-free RL algorithms have difficulty converging.

Reda et al. (2020) explored the effect of key environmental components on RL policy, considering TD3 algorithm performance variations across different Initial-state Distribution, State Representation, Control Frequency, and Choice of Action Space. Their experiments with PyBullet environment reward functions, particularly the Survival Bonus differences, demonstrated that specific components in the reward function provide a critical form of reward shaping that can significantly affect agent behavior.

Jayawardana et al. (2022) placed more emphasis on the importance of evaluating RL policy in more diverse environments. They explored the impact of task under specification on RL methods, noting that a family of RL tasks typically corresponds to hundreds or thousands of MDP instances with varying parameters rather than individual point MDPs, and found experimentally that some DRL methods reported to be superior to traditional control methods

under the traffic signal control task performed significantly worse when evaluated across a range of MDP environments rather than individual MDP environments. Wolgast and Nieße (2024) focused on the practical problem of optimal power flow, implementing 13 different design variants of environments and comparing the performance of RL training in these environments. When designing different environments based on data sampling distribution and evaluating them, they found that training with realistic time-series data was significantly better than training with randomly sampled data.

Bono et al. (2025) challenged the traditional assumption that RL methods perform optimally when the training and testing environments are consistent. They created 60 environment variants based on the ATARI game by means of noise injection or semantic modification and found that the RL agent performs better in noisy environments if it is trained in a noiseless "indoor" environment.

4 Reinforcement Learning Preliminaries

This section contains foundational knowledge and key concepts of reinforcement learning, which is the basis for the research methodology in Section 5. General concepts, elements, Bellman equation, and the deep reinforcement learning (DRL).

4.1 General Concepts and Markov Decision Process

Reinforcement learning represents a branch of machine learning that learns "what to do" through trial and error, by evaluating rewards and improving actions taking based on those evaluations (Sutton and Barto, 2018). Unlike supervised learning, which relies on labeled datasets provided in advance to solve tasks such as classification, regression, or ranking, reinforcement learning generates training data through interaction between the agent and the environment: As shown in Figure 3, at time step t, the agent is in state S_t, takes action A_t, applies it to the environment, and receives a reward R_t from the environment. The environment then changes to the next state S_{t+1}, the value-based method and the policy-based method.

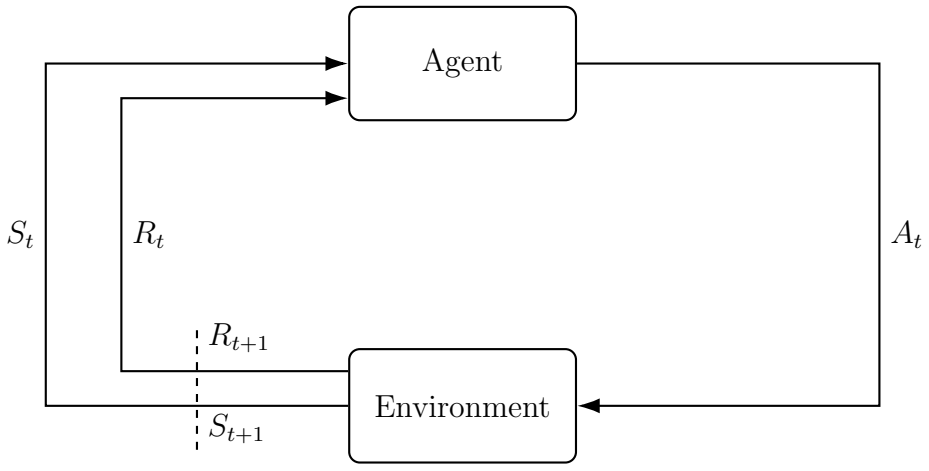


Figure 3: *Agent-environment Dynamics in RL.* (Sutton and Barto, 2018)

The state S_t, action A_t, and reward R_t are key components of a Markov Decision Process (MDP), which is used to model the environment. An MDP is defined as a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$,

where: (1) $\mathcal{S}$ is the set of possible states (state space). (2) $\mathcal{A}$ is the set of available actions in the environment (action space). (3) $\mathcal{P}$ is the set of environmental dynamics $p(s'|s,a)$, that is, probability of transitioning from state s to state s' given that action a is taken. In deterministic environments, $p(s'|s,a) = 1$. (4) $\mathcal{R}$ represents the collection of immediate rewards $r(s'|(s,a))$ received by the agent from taking action a from state s to the next state s'.

4.2 Elements of Reinforcement Learning

Reinforcement learning's important elements include the agent's policy, the environment's reward signal, and the value function of state-action pairs.

The policy π defines how the agent selects actions based on its observation (which may be a subset of the environment's full state $\mathcal{S}$ that the agent can perceive). A policy can be deterministic, meaning the agent always selects a certain action given a specific state; or stochastic, meaning the agent chooses among possible actions with certain probabilities.

Reward signals are the feedback provided by the environment when an agent's specific state-action pair acts upon it. The ultimate objective of reinforcement learning is to maximize the sum of the rewards, rather than the reward at a single step, because an action yielding low reward may lead to a set of highly rewarding states in the future. In this context, a value function, Q-function, is used to describe the goodness of a certain state (or state-action pair), that is, the expected sum of future rewards (discounted by γ) from that point until the end of the episode. For a MDP starting at state s under policy π, the value function is defined as shown in Equation (2). Similarly, for the action value under a specific state-action pair, the definition is given in Equation (3). How to accurately estimate the value function is a critical aspect in reinforcement learning.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right], \quad \text{for all } s \in \mathcal{S}, \tag{2}$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right] \tag{3}$$

During the training process, we generally encourage the agent to exploit its current knowledge by choosing actions with higher values. However, to avoid falling into the local optimum, it's also necessary for the agent to take unseen actions, in hopes of discovering better outcomes. The goal is to strike a balance between exploitation and exploration to achieve a most effective policy overall.

4.3 Bellman Equation

With the definition of the value function in Equation (2):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned} \tag{4}$$

Equation (4) is the basic form of the Bellman equation (Bellman, 1966). It reveals the relationship between the value of the current state and that of the next state, and also shows that the value function can be computed through an iterative process.

4.4 Value-based and Policy-based Methods

One approach to determining the optimal policy is the value-based method, which involves evaluating the Q-values of each action and then selecting actions maximizing these Q-values. Q-learning (Watkins and Dayan, 1992) is a tabular value-based method based on the Bellman equation, which maintains a Q-value table of state-action pairs and updates it through equation (5) until convergence. Here, α is the learning rate.

$$Q^{\text{new}}(S_t, A_t) \leftarrow (1-\alpha) \cdot Q(S_t, A_t) + \alpha \cdot \left(R_{t+1} + \gamma \cdot \max_a Q(S_{t+1}, a)\right) \tag{5}$$

However, tabular Q-learning is not practical when facing problems with large state or action spaces, nor can it generalize to states that the agent has never encountered. To address, deep reinforcement learning uses neural networks to approximate Q-values, which is known as deep-Q learning (DQN) (Mnih et al., 2013).

However, DQN is only applicable to discrete action spaces. For continuous action spaces, policy-based methods such as REINFORCE (Williams, 1992) are used, which optimize the policy directly through an objective function. For a policy π_θ parameterized by θ, we can define the objective function $J(\theta)$ as the expected long-term return. Taking the gradient of $J(\theta)$ with respect to theta can obtain equation (6), then the optimal theta can be found by maximizing this gradient through gradient ascent.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \left[\sum_{t \in 0:T} \nabla_\theta \log \pi_\theta(A_{t,n} \mid S_{t,n}) \sum_{\tau \in t:T} (\gamma^\tau R_{\tau,n}) \right] \tag{6}$$

5 Methodology

This section describes the methodologies and tools used in this research, including the design of a stowage planning environment compatible with the OpenAI Gym interface (Brockman et al., 2016) for training RL agents, the integration of the stable-baseline3 RL algorithm library (Raffin et al., 2021), employed algorithms, the application of action masks, and the training procedures.

5.1 Stowage Planning Gym Environment

This research developed a stowage planning gym environment (SPGE) to address data scarcity issues, as well as the challenge that stowage planning simulation environments are often highly coupled with vessel structures and container types, resulting in excessive complexity. This environment is a high-level abstraction of the terminal stowage process, representing the vessel and yard as cubes, and treating individual container slots in the vessel and yard as operational units. The attributes of these units (e.g., coordinates and occupancy status) is the basis for both simplified constraint enforcement according to vessel planning rules and the construction of the state space. These attributes include:

1. Slot Coordinates. These are (bay, row, tier) triplets indicating the position of a slot on the vessel or yard. Each slot has a unique coordinate.
2. Occupancy. This indicates whether the slot contains containers or not. For a yard slot, this value being set to 1 means it holds a container waiting to be loaded, while for a vessel slot, 0 means it still needs to be filled.
3. Group. This value indicates either the expected container type for a vessel slot, or the actual container type at a yard slot. It is used for abstracting vessel planning rules that require certain containers to be stowed in specific vessel areas (e.g., refrigerated containers need to be placed in powered sections of the ship). A slot on the vessel can only be filled with container with the same group value.

SPGE allows problem scale control by a config dictionary during environment initialization. The config dictionary will specify the number of bays, rows, and tiers for the ship and yard, as well as the number of containers to be loaded and the number of groups. In addition to the basic stowage planning gym environment, where the agent controls the container loading sequence, this study also includes a variant where the agent jointly controls both containers and cranes, the stowage planning gym environment with multiple cranes (SPGE-MC).

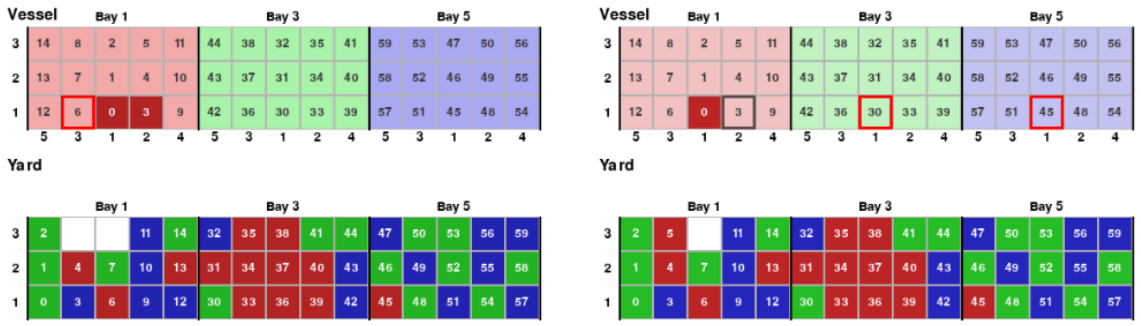
5.1.1 Basic SPGE

Based on a provided config dictionary, the environment will generate a vessel and yard of specified sizes, along with containers to be loaded with specified numbers, and randomly assign these containers to groups, ensuring that scenarios can be reproduced by setting a seed in the environment configuration. For a vessel or yard of size $(n_{\text{Bay}}, n_{\text{Row}}, n_{\text{Tier}})$, the state is represented as a matrix with $n_{\text{Bay}} \times n_{\text{Row}} \times n_{\text{Tier}}$ rows and n_{SlotAttr} columns, where each row corresponds to a unique slot and its associated attributes.

Figure 4a shows the visualization of an example environment with 3 bays, 5 rows, 3 tiers in both vessel and yard, and 45 containers. The corresponding vessel and yard state data can be found in Table 1.

SPGE uses a sequencer to determine the order of vessel slot to be filled in order to satisfy constraint 4 in Section 2.3.1. At each step, for the vessel slot specified by the sequencer, the number of actions can be taken equals the total number of yard slots. The observation is formed by concatenating and flattening the attribute values of the vessel state, yard state, and the target vessel slot.

Therefore, for an environment having a vessel of size $(n_{\text{VBay}}, n_{\text{VRow}}, n_{\text{VTier}})$, a yard of size



(a) SPGE (b) SPGE-MC

Figure 4: *Visualization of SPGE and SPGE-MC.* Cross-section views of vessel and yard bays. The upper and lower parts of the figure represent the current states of the vessel and yard slots, respectively. Each square represents a container slot, while different colors indicate different groups. Light-color squares in vessel requires to be filled with containers with corresponding group, while white squares in yard are empty slots. Numbers on squares are unique slot IDs, while numbers below and left indicate rows and tiers. In (a)(b), outlined squares are target vessel slots to be filled. The difference is that in (b), multiple sequencers correspond to the number of cranes, and a time mechanism is introduced, which brings crane availability into consideration. A red highlight indicates that the crane associated with the sequencer is idle and the vessel slot can be filled at this step, while gray highlight indicates the slot cannot be filled due to unavailability of the crane.

$(n_{\mathsf{YBay}}, n_{\mathsf{YRow}}, n_{\mathsf{YTier}})$, n_{Group} kinds of containers with number of $n_{\mathsf{C}}$to be loaded, we define:

$$n_{\mathsf{V}} = n_{\mathsf{VBay}} \times n_{\mathsf{VRow}} \times n_{\mathsf{VTier}} \tag{7}$$

$$n_{\mathsf{Y}} = n_{\mathsf{YBay}} \times n_{\mathsf{YRow}} \times n_{\mathsf{YTier}} \tag{8}$$

such that $n_{\mathsf{C}} \leq n_{\mathsf{Y}}$.

The observation space of SPGE will be represented as a one-dimensional vector of size $(n_{\mathsf{V}} + n_{\mathsf{Y}} + 1) \times n_{\mathsf{SlotAttr}}$, while the discrete action space has a size of n_{Y}.

For state transitions, there are two cases to consider. If the selected action is invalid, such as selecting a yard slot with no container, or selecting a container whose group does not match the target vessel slot, no changes occur in either the vessel or yard state. In this case, a large negative reward (-100) is returned as a penalty, since the transition is not valid.

If the selected action is valid, the environment calculates the number of containers stacked above the selected one in the yard as shifter, which will be returned as a negative reward. The container's attribute information is then copied from the yard slot to the target vessel slot. Meanwhile, all containers in the yard above the selected one (i.e., those in the same bay and row but with higher tier values) are shifted down by one tier. For example, in Figure 4a, if container 33 in the yard is selected, then after the shift, containers 34 and 35 will have their tier values changed from 2 and 3 to 1 and 2, respectively. The sequencer will also advance to

Table 1: *Partial observation values of vessel and yard corresponding to Figure 4a.*

(a) Vessel State

Slot Number	Bay	Row	Tier	Occupancy	Group
0	1	1	1	1	0
3	1	2	1	1	0
6	1	3	1	0	0
30	3	1	1	0	1
...	...	...	...	...	...

(b) Yard State

Slot Number	Bay	Row	Tier	Occupancy	Group
0	1	1	1	1	1
1	1	1	2	1	1
4	1	2	2	1	0
5	1	2	3	0	-1[1]
...	...	...	...	...	...

[1] Since this yard slot has no container, the container attribute *group* does not apply.

the next vessel slot that needs to be filled. When there's no container in the yard or all vessel slots are filled, the environment terminates.

5.1.2 LME

We also use an industrial simulator and training environment from Loadmaster.ai[1] to conduct experiments. This environment includes real-world yard and vessel data from Rotterdam Shortsea Terminal (RST). The Loadmaster training environment (LME) operates on a similar principle to the Basic SPGE described in Section 5.1.1, but its sequencer of the vessel incorporates more complex rules to satisfy the problem constraints mentioned in Section 2.3.1. Furthermore, LME also has more complex slot attributes than Basic SPGE. In addition to coordinates and occupancy status, these attributes also include information such as container weight, ISO classification, dangerous goods (IMO) status, and Reefer status (requiring power grid connection).

5.1.3 SPGE-MC

SPGE-MC is an extension of the basic SPGE that allows joint control over both container selection and crane assignment, while basic SPGE can be regarded as a particular case of SPGE-MC with only one single crane. In this environment, a single action is represented as a (container, crane) pair, while action validity is determined based on a global clock and estimated operation time.

To simplify the problem, SPGE-MC assumes that the stowage operation duration for each container is independent. At environment initialization, SPGE-MC generate n_{C} random values as operation times for all containers, while also using seed for reproducing scenarios.

[1] https://loadmaster.ai/

Consider a stowage environment with a vessel and a yard of sizes n_{V} and n_{Y} respectively (as defined in Equations (7) and (8)), involving n_{C} containers to be loaded and n_{CR} cranes available for scheduling. To track the crane availability, SPGE-MC maintains a global clock t and a vector $\boldsymbol{\tau} = [\tau_1, \tau_2, \ldots, \tau_{n_{\mathsf{CR}}}]$. Here, t denotes the duration since the beginning of the stowage operation, while each τ_i denotes the time (relative to the start of operation) after which crane i becomes available again. Therefore, we can get the feasible (container, crane) pair at each step by inspecting the non-negative elements of $\boldsymbol{\tau} - t$ and the valid containers according to section 5.1.1. If there are no non-negative elements in $\boldsymbol{\tau} - t$ after executing an action at a certain step, t will be advanced to the earliest time at which any operating crane becomes available.

SPGE-MC divides all vessel bays into n_{CR} bay zones, each operated by an assigned crane with a corresponding sequencer to prevent crane interference. When a crane completes its loading task within a zone, it is scheduled to take over a bay that still requires loading but is not currently operated by any crane. Figure 4b shows an SPGE-MC environment with the same yard and vessel sizes, number of containers, and group numbers as in Figure 4a, but with 3 cranes, which is shown by the 3 outlined squares in the vessel.

For the observation space, SPGE-MC contains richer information than SPGE. In addition to the vessel state and yard state in Table 1, it also includes the crane availability state and the crane operating state, which records which container each crane is currently handling at a certain step. For example, for a given step in Figure 4b, the corresponding crane availability state and crane operating state, as well as sequencer state are shown in Table 2:

Table 2: *Crane status related observation values corresponding to Figure 4b.* The three highlighted squares from left to right correspond to Crane 1, 2, and 3. The size of the state is determined by n_{CR}.

(a) Crane Availability State

Crane 1	Crane 2	Crane 3
100[1]	0	0

(b) Crane Operating State

Crane 1	Crane 2	Crane 3
8	-1[2]	-1

(c) Crane Sequencer State

Crane 1	Crane 2	Crane 3
3	30	45

[1] Indicates that the crane will be available in 100 seconds.

[2] Indicates crane not currently loading a container.

The observation space of SPGE-MC is composed of the flattened concatenation of the basic SPGE observation, the crane availability state, the crane operating state, and the crane sequencer state, which is a vector of size $(n_{\mathsf{V}} + n_{\mathsf{Y}} + 1) \times n_{\mathsf{SlotAttr}} + n_{\mathsf{CR}} \times 3$. The action space is a discrete space of size $n_{\mathsf{C}} \times n_{\mathsf{CR}}$.

For state transitions, if an invalid action is taken (either due to the same reasons as in the basic

SPGE or because the selected crane is unavailable), none of the vessel, yard, or crane-related states are updated. If the action is valid, the yard and vessel states are updated to reflect the stowage of the selected container, while the crane-related states are also changed accordingly, such as updating the corresponding elements in $\boldsymbol{\tau}$, advancing t if necessary.

5.1.4 SPAEC

Another way to combine crane scheduling with container selection is to make container selection decisions in a multi-agent environment, with each agent modeled as an individual crane. Due to the homogeneity of the decision making in this environment, we can consider using a single policy network to act as multiple agents. To achieve this, we developed the SPAEC (Stowage Planning Agent Environment Cycle) environment, compatible with the PettingZoo interface (Terry et al., 2021), to support training with a single policy in a multi-agent setting.

SPAEC has the same action space as the basic SPGE but adopts the same observation space as SPGE-MC (as shown in Table 2) and similar time advancement logic to ensure that each crane can only perform container selection when available. Unlike SPGE-MC, SPAEC does not couple the (container, crane) pairs into the action space. Instead, crane scheduling is implicitly handled through agent selection. The availability of an agent at a given timestep still depends on the current time t and the availability array $\boldsymbol{\tau}$.

Additionally, when there are multiple cranes available at the same time t, SPGE-MC allows the agent to freely choose any crane, while SPAEC has a fixed rotation order for agent selection. For example, in the situation of Figure 4b, SPGE-MC allows agent to schedule either crane in bay 3 or bay 5 first. But in SPAEC, only crane at bay 3 will be scheduled first.

5.2 Stable Baselines 3

This study uses algorithms implemented in the open source reinforcement learning framework Stable-Baseline3 for comparative experiments. SB3 is an improved version of OpenAI Baselines, built on the PyTorch (Paszke et al., 2019) backend, and integrates a variety of on-policy and off-policy algorithms. It includes implementations of DRL algorithms such as A2C, DQN, and PPO, supporting both discrete and continuous action spaces, focusing on simplicity, reliability, and reproducibility of experiments.

SB3 offers comprehensive documentation, practical usage examples, and an easy-to-use API. It also supports integration with Weights & Biases (wandb) (Biewald, 2020) for experiment tracking and visualization. In addition, SB3 provides well-tuned default hyperparameters generalizing well across various environments, which helps fair comparison among algorithms.

5.3 Algorithms

In this study, we compare the performance of various reinforcement learning algorithms within the SPGE-MC and SPAEC environments. We focus on algorithms that support discrete action spaces, including the value-based method Deep Q-Network (DQN), distributional method Quantile Regression Deep Q-Network (QR-DQN) and actor-critic family algorithms such as Advantage Actor-Critic (A2C), Trust Region Policy Optimization (TRPO), and Proximal Policy Optimization (PPO). This section will provide an overview of the underlying principles of these algorithms.

5.3.1 DQN

Deep Q-Network (DQN) is an off-policy algorithm proposed by the DeepMind team as a variant of Q-learning (Mnih et al., 2013). As a value-based method, it learns good policies by encouraging the agent to select actions that could maximize future rewards, supporting exploration-exploitation strategies such as ϵ-greedy and Boltzmann exploration. DQN uses neural networks, known as the "Q-network", to approximate the action-value function. Q-network can be trained by minimizing the loss function (as shown in equation (9)) at each iteration.

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(\mathbb{E}_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \mid s, a\right] - Q(s, a; \theta_i)\right)^2\right] \tag{9}$$

While learning the policy, DQN selects the action with the highest Q-value based on the current parameters (i.e., a greedy policy), but actions taken in the environment are determined according to the exploration strategy. To address data correlation and non-stationarity, DQN uses an experience replay buffer which stores state transitions (s_t, a_t, r_t, s_{t+1}) at each timestep and randomly samples them for training the Q-network.

To improve stability, DQN maintains two Q-networks: an evaluation network and a target network. The target network is used to compute the maximum Q-value for the next state (i.e., the $\gamma \max_{a'} Q(s', a'; \theta_{i-1})$ part), while the evaluation network calculates the Q-value of the current action and is involved in gradient computation and real-time weight updates. Periodically, the evaluation network's parameters are used to refresh those of the target network, which allows a more stable learning target.

Algorithm 1 Deep Q-learning with Experience Replay (Mnih et al., 2013)

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with random weights $\theta^- = \theta$
for episode = 1 to M **do**
    for $t = 1$ to T **do**
        if random number $< \varepsilon$ **then**
            Select a random action a_t
        else
            Select $a_t = \arg\max_a Q(s_t, a; \theta)$
        end if
        Execute action a_t in emulator and observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in D
        Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameter θ
        Every C steps reset $\hat{Q} = Q$
    end for
end for

5.3.2 QR-DQN

Quantile Regression Deep-Q Network (QR-DQN) is an algorithm proposed based on Categorical DQN (C51)'s (Bellemare et al., 2017) improvement over DQN, incorporating the idea of quantile regression. (Dabney et al., 2018) Like C51, it predicts the value distribution corresponding to the state-action pair rather than just predicting the expected value, applying the approximate value distribution to the Bellman equation to address the inherent uncertainty in reward outcomes.

It addresses a gap between the theory and practice of distributional reinforcement learning in the C51 algorithm, which is unable to apply the Wasserstein distance, a desirable metric for measuring distributional differences, in gradient descent and network training.

The general structure of QR-DQN is similar to DQN: it uses an evaluation network, a target network, and experience replay for training. Unlike DQN, whose Q-network outputs a single Q-value as the expected value for a state-action pair, QR-DQN outputs a set of quantiles to approximate the full distribution of Q-values. In terms of the loss function, QR-DQN does not use the pinball function (as shown in equation (10)) commonly used in quantile regression, because pinball function is not differentiable at zero. It adopts quantile Huber loss instead, which is smooth and differentiable around zero.

$$\mathcal{L}_\tau(u) = \begin{cases} \tau u, & \text{if } u \geq 0 \\ (\tau - 1)u, & \text{if } u < 0 \end{cases} \tag{10}$$

$$\mathcal{L}_\kappa(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \leq \kappa \\ \kappa\left(|u| - \frac{1}{2}\kappa\right), & \text{otherwise} \end{cases} \tag{11}$$

$$\rho_\tau^\kappa(u) = |\tau - \delta_{\{u<0\}}| \cdot \mathcal{L}_\kappa(u) \tag{12}$$

The quantile Huber loss is shown in equation (12), where $\tau \in (0, 1)$ is the quantile, and κ is the smoothing parameter of the Huber loss. It controls the range around $u = 0$ within which the Huber loss is applied, and outside of which the standard quantile loss is used. $\delta_{\{u<0\}}$ denotes an indicator function that equals 1 when $u < 0$, and 0 otherwise. By minimizing the quantile Huber loss, QR-DQN aligns the predicted distribution from the evaluation network with the target distribution from the target network in terms of the Wasserstein metric. When selecting actions, the agent still chooses the action with the highest expected Q-value.

5.3.3 A2C

Advantage Actor-Critic (A2C) (Mnih et al., 2016) is a RL algorithm based on the actor-critic architecture. The actor-critic architecture is a combination of the policy-based and value-based methods described in Section 4.4, incorporating both a policy network (actor) and a state-value network (critic).

A2C uses an N-step bootstrap approach to estimate the returns for each time step. During a single episode, it samples N consecutive steps $(s_t, a_t, r_t, s_{t+1}), \ldots, (s_{t+N-1}, a_{t+N-1}, r_{t+N-1}, s_{t+N})$ from the environment following its policy, and calculates the return at each step by summing the rewards received from t to $t + N - 1$, bootstrapping from the estimated value $V(s_{t+N})$ provided by the critic network, to approximate the action-value function $Q^\pi(s_t, a_t)$. If the episode terminated at s_{t+N}, $V(s_{t+N})$ will be 0.

Algorithm 2 Advantage Actor-Critic (A2C) (Mnih et al., 2016)

Initialize global step counter $T \leftarrow 0$
Initialize global parameters θ, θ_v
while $T < T_{\max}$ **do**
    Reset gradients: $d\theta \leftarrow 0$, $d\theta_v \leftarrow 0$
    Synchronize local parameters: $\theta' \leftarrow \theta$, $\theta'_v \leftarrow \theta_v$
    Get initial state s_t
    $t_{\text{start}} \leftarrow t$
    repeat
        Select action $a_t \sim \pi(a_t|s_t;\theta')$
        Execute a_t in environment
        Observe reward r_t and next state s_{t+1}
        $t \leftarrow t+1$, $T \leftarrow T+1$
    until terminal s_t or $t - t_{\text{start}} == t_{\max}$
    if terminal s_t **then**
        $R \leftarrow 0$
    else
        $R \leftarrow V(s_t;\theta'_v)$ ▷ Bootstrap from value function
    end if
    for $i = t-1, \ldots, t_{\text{start}}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients:
            $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta') \cdot (R - V(s_i;\theta'_v))$
            $d\theta_v \leftarrow d\theta_v + \nabla_{\theta'_v} \frac{1}{2}(R - V(s_i;\theta'_v))^2$
    end for
    Perform gradient update on θ and θ_v using $d\theta$, $d\theta_v$
end while

In addition, A2C introduces the advantage function $A_t = Q^\pi(s_t, a_t) - V(s_t)$, which measures the relative benefit of an action over the expected value of a state. During training, A2C not only maximizes the advantage-weighted policy gradient $\mathcal{L}_{\text{policy}} = \mathbb{E}_{(s_t,a_t)}[\log \pi_\theta(a_t|s_t) \cdot A_t]$, but also minimizes the temporal-difference (TD) loss between the N-step return R_t and the critic's estimate $V(s_t)$ to reduce the training variance, defined as:

$$\mathcal{L}_{\text{value}} = \frac{1}{2} \sum_t (R_t - V(s_t; \theta_v))^2 \tag{13}$$

Where $R_t = \sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N V(s_{t+N})$. In the mean time, A2C introduces a policy entropy regularization term $\mathcal{L}_{\text{entropy}} = -\mathbb{E}_{s_t}[\mathcal{H}(\pi_\theta(s_t))]$ to encourage agent exploration, where $\mathcal{H}$ is the entropy of policy distribution.

Thus, the overall loss function is denoted as:

$$\mathcal{L}_{\text{total}} = -\mathcal{L}_{\text{policy}} + \alpha \cdot \mathcal{L}_{\text{value}} + \beta \cdot \mathcal{L}_{\text{entropy}} \tag{14}$$

where alpha and beta are weights of value loss term and entropy loss term respectively. Algorithm 2 describes the training process of A2C.

5.3.4 TRPO

Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) is an application of numerical optimization algorithms in reinforcement learning. As an on-policy algorithm, it differs from traditional policy gradient methods, and uses a proximity-maximize loop based on the idea of trust region optimization instead of stochastic gradient ascent to optimize the policy, allowing for more stable policy updates and avoiding policy collapse.

The idea of trust region optimization is as follows: for an objective function $J(\theta)$ that is difficult to evaluate or differentiate, to solve $\arg\max_\theta J(\theta)$, we can start with an arbitrary value θ_{old}. For θ_{old}, we can define a neighborhood $\mathcal{N}(\theta_{\text{old}})$ and construct a simpler surrogate function $\mathcal{L}(\theta)$ which approximates $J(\theta)$ well within that neighborhood. If such a neighborhood exists, then $\mathcal{N}(\theta_{\text{old}})$ is called the trust region of θ_{old}. We then solve $\arg\max_\theta \mathcal{L}(\theta)$ within $\mathcal{N}(\theta_{\text{old}})$, and the resulting update improves $J(\theta)$ compared to $J(\theta_{\text{old}})$. After updating θ_{old}, repeat the surrogate construction and maximization, and iterate until convergence.

In reinforcement learning, we hope to solve the objective function $\mathbb{E}_s[V_\pi(s)]$, that is, optimize the expected state value given the policy. Since $V_\pi(s)$ depends on the policy parameter θ, we can expand it as:

$$\begin{aligned} V_\pi(s) &= \sum_a \pi(a|s;\theta) Q_\pi(s,a) \\ &= \sum_a \pi(a|s;\theta_{\text{old}}) \cdot \frac{\pi(a|s;\theta)}{\pi(a|s;\theta_{\text{old}})} Q_\pi(s,a) \\ &= \mathbb{E}_{a\sim\pi(\cdot|s;\theta_{\text{old}})} \left[\frac{\pi(a|s;\theta)}{\pi(a|s;\theta_{\text{old}})} Q_\pi(s,a) \right] \end{aligned} \tag{15}$$

Thus, to optimize $J(\theta)$, we can use the surrogate function $\mathcal{L}(\theta)$:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}\frac{\pi(a_i|s_i;\theta)}{\pi(a_i|s_i;\theta_{\text{old}})}Q_\pi(s_i,a_i) \tag{16}$$

where $Q_\pi(s,a)$ can be estimated using n-step discounted rewards from sampled trajectories. In other words, after sampling n steps from the environment, we formulate the problem as:

$$\begin{aligned}\theta_{\text{new}} &= \arg\max_\theta \tilde{L}(\theta \mid \theta_{\text{old}}) \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{\pi(a_i \mid s_i;\theta)}{\pi(a_i \mid s_i;\theta_{\text{old}})}\left(r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + \cdots + \gamma^{n-i} r_n\right)\end{aligned} \tag{17}$$

With the constraint that θ stay within $\mathcal{N}(\theta_{\text{old}})$. This trust region can be enforced by measuring the difference between the new policy π_θ and old policy $\pi_{\theta_{\text{old}}}$, using the Kullback–Leibler (KL) divergence (Kullback and Leibler, 1951) as the distance metric since both policies are parameterized distributions. The final constrained optimization becomes:

$$\theta_{\text{new}} \leftarrow \arg\max_\theta \tilde{L}(\theta|\theta_{\text{old}}), \quad \text{s.t.} \quad \frac{1}{n}\sum_{i=1}^{n}\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_i)\|\pi_\theta(\cdot|s_i)] < \delta \tag{18}$$

which can be solved by numerical optimization techniques.

5.3.5 PPO

TRPO can achieve a stable convergence theoretically. However, since the constraint problem under KL divergence in equation (18) involves complex quadratic programming, TRPO computationally intensive and complex to implement. To address this, Proximal Policy Optimization (PPO) (Schulman et al., 2017) was proposed as a simplified method that inherits the core idea of TRPO, aiming to strike a balance between implementation complexity and algorithmic stability. It transforms the trust region in the TRPO algorithm into a gradient clipping mechanism to limit the step size of the policy update. It uses the surrogate loss function $L(s,a,\theta_k,\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)$ to approximate the objective function $J(\theta)$, and applies gradient updates to $L(s,a,\theta_k,\theta)$ in the following way:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)}A^{\pi_{\theta_k}}(s,a),\ g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right) \tag{19}$$

where $g(\epsilon, A) = \begin{cases}(1+\epsilon)A & \text{if } A \geq 0 \\ (1-\epsilon)A & \text{if } A < 0\end{cases}$. ϵ is a hyperparameter that limits the step size of a single policy update. This clipping means that if the new policy increases the probability of action a too much because it performs better, the agent will not receive a better reward if the increase exceeds $1+\epsilon$. Conversely, if the advantage function is negative, the agent also won't benefit from drastically decreasing the probability of the corresponding action.

Algorithm 3 describes the training process of PPO, where the optimization objective is:

$$L^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{20}$$

Here, c_1, c_2 are coefficients, S denotes an entropy bonus term, and $L_t^{\text{VF}}(\theta)$ denotes the value function loss.

Algorithm 3 PPO, Actor-Critic Style (Schulman et al., 2017)

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L w.r.t. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

5.4 Action Masks

In the experimental environments, due to constraints such as the same group numbers, slot occupancy, and crane availability, there are often a large number of invalid actions in the action space at any given time. A straightforward way to prevent the agent from selecting invalid actions is to make the environment state unchanged and return a large negative reward when the agent selects an invalid action. However, in practice, this naïve approach makes the training process difficult to converge, especially when the action space is large and valid actions take a small portion. The agent often gets trapped in invalid exploration most of the time, causing the policy fail to improve.

Alternative approach is to use action masking, which filters out invalid actions at the policy network level to prevent the agent from selecting them. In discrete action spaces, a common way to apply action masks for policy-based methods is to set the logits of invalid actions to negative infinity, which allows zero probability of those actions after applying softmax function. Similarly, for value-based algorithms such as DQN, the Q-values of invalid actions in the network are set to negative infinity, ensuring that the policy does not select invalid actions.

Huang and Ontañón (2020) compared action masking and the naïve reward penalty approach in μRTS environments (Huang et al., 2021) of small size map and larger size map. Their experiments showed that the naïve reward penalty approach only converges in small-scale problems, while action masking not only converges faster but also scales better as the problem size grows.

This study has similar findings through the experiment shown in Figure 5. For a very simple scenario with 4 containers, PPO without action masking still requires 28,000 timesteps to fit, whereas PPO with action masking reaches optimal performance right after the first rollout (note that the episode mean reward is only recorded after the first rollout is completed, which is why data points only appear on the graph when timesteps > 0). In a slightly more complex scenario with 64 containers, PPO with action masking starts fitting at around 80,000 timesteps. In contrast, training without action masking is significantly slower, and the episode mean reward remains low, which is because the algorithm wastes a large portion of the training budget exploring invalid actions.

For effective training, we modified algorithms in Stable-Baselines3 to apply action masking. SB3 already provides an implementation of PPO with action masking and an actor-critic policy network with built-in filter support. When modifying A2C and TRPO algorithms, we used the actor-critic policy network with filter support from SB3, along with a rollout buffer capable of storing action masks corresponding to states. For DQN, we added an action mask filtering

(a) PPO without Action Mask (Small Scale: 2x2x1, 4C)

(b) PPO with Action Mask (Small Scale: 2x2x1, 4C)

(c) PPO without Action Mask (Large Scale: 4x4x4, 64C)

(d) PPO with Action Mask (Large Scale: 4x4x4, 64C)

Figure 5: *Training Curves of PPO with and without action masking across different problem scales.* (a) and (b) illustrate performance on a small scale (Vessel/Yard: 2x2x1, 4 Containers, 3 Types). (c) and (d) illustrate performance on a larger scale (Vessel/Yard: 4x4x4, 64 Containers, 3 Types). Action masking is applied in (b) and (d).

layer on top of the Q-network and modified the replay buffer to store action mask information.

5.5 Experimental Setup

This section will describe the experimental setup covering setting up the environment configurations for various research subproblems, the setting of hyper-parameter tuning and training, and the experimental infrastructure.

5.5.1 Scenario Setup

We designed a series of problem scenarios to address the three research questions proposed in Section 1, gradually increasing the complexity of the problem in terms of vessel and yard size, container types, and number of cranes. Corresponding experiments were conducted in the Basic SPGE, SPGE-MC, and SPAEC. These scenarios include:

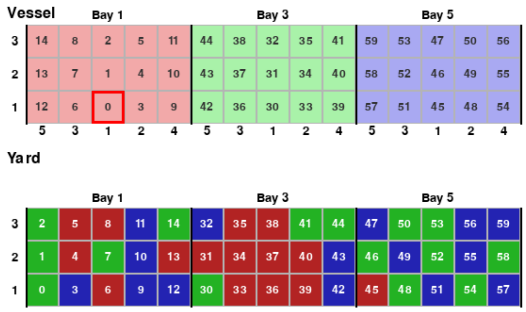
1. Both the vessel and yard have dimensions of 3 bays, 5 rows, and 3 tiers (totaling 45 slots), and 45 containers need to be stowed, with 3 container types. One single crane.
2. Same vessel and yard size as Scenario 1, with 45 containers to stow, but with 8 container types. One single crane.
3. The vessel size is 3 bays, 5 rows, 3 tiers; the yard size is 8 bays, 5 rows, 5 tiers (200 yard slots), and 45 containers need to be stowed, with 8 container types. One single crane.
4. The vessel size is 8 bays, 5 rows, 5 tiers; the yard is 3 bays, 5 rows, 3 tiers (200 vessel slots), and 45 containers need to be stowed, with 8 container types. One single crane.
5. Both vessel and yard have 8 bays, 5 rows, 5 tiers (200 vessel slots), and 200 containers need to be stowed, with 8 container types. One single crane.
6. Same vessel and yard sizes as Scenario 3, with 3 cranes.
7. Same vessel and yard sizes as Scenario 5, with 3 cranes.
8. Same vessel and yard sizes as Scenario 5, with 5 cranes.

Figure 6 shows the corresponding problem visualizations for these eight scenarios. Specifically, Figure 6a corresponds to Scenario 1, 6b to Scenario 2, 6c to Scenarios 3 and 6, 6d to Scenario 4, and 6e to Scenarios 5, 7, and 8.

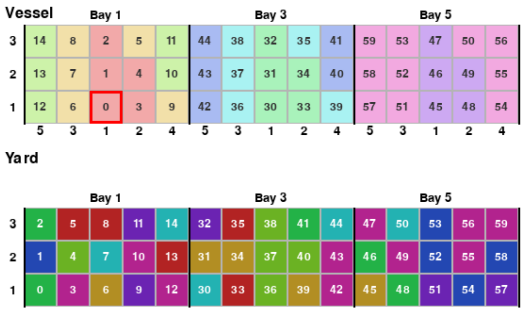
Scenarios 1-5 will be used for the SPGE experiment. Among them, Scenario 1 and Scenario 4 are relatively easier problems to solve. Scenario 1 has the fewest number of containers and types to consider. In Scenario 4, due to the sequencer's characteristic of filling bays in ascending order, the container types in the stowage sequence are not interleaved as much as in Scenario 2. Furthermore, the smaller number of containers to be loaded means this scenario does not require heavy reliance on long-term planning, unlike Scenario 5.

Scenario 6-8 will be used for SPGE-MC and SPAEC to conduct comparative experiments for single agent and multi-agent formulation.

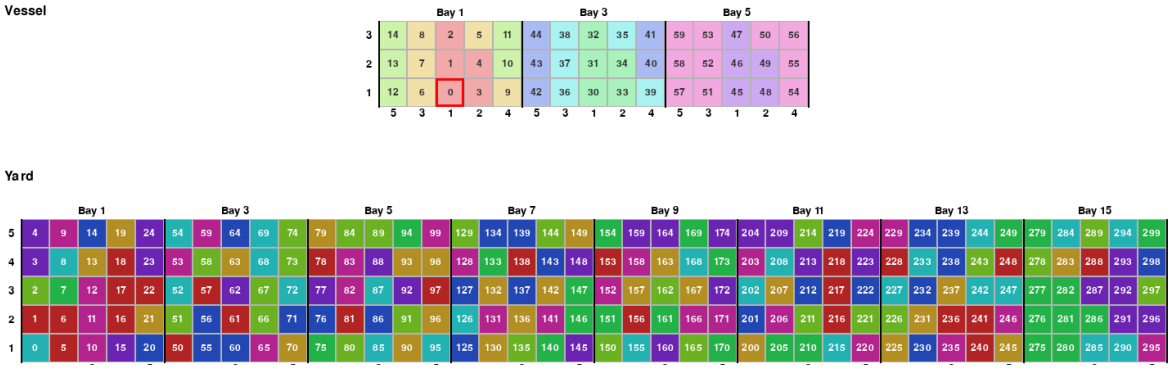
For the LME, we conducted experiments using the same vessel across three scenarios: A, B, and C. These scenarios contain 100, 245, and 642 containers to be stowed, respectively, simulating operations of different scales.



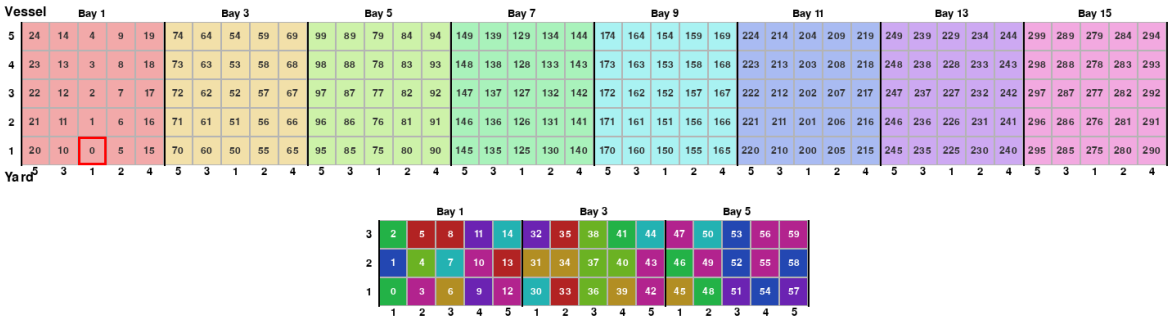
(a) Scen. 1: V/Y 3×5×3 (45 slots ea.), 3 types



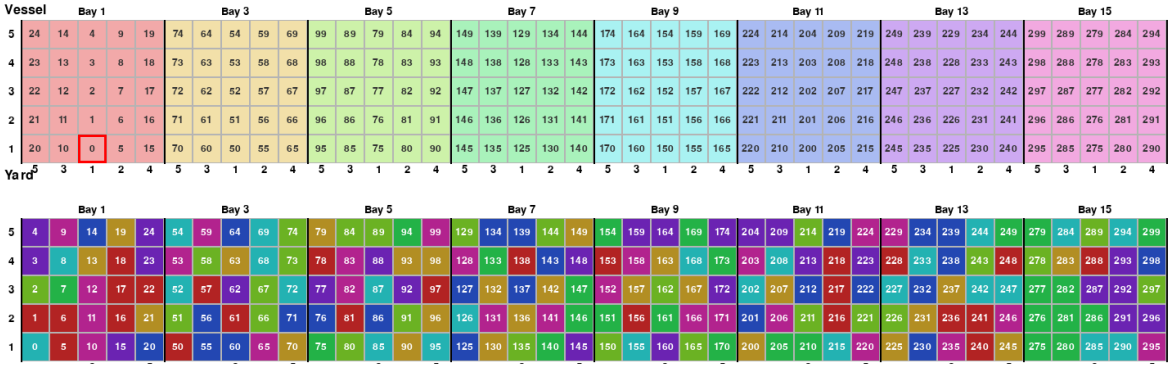
(b) Scen. 2: V/Y 3×5×3 (45 slots ea.), 8 types



(c) Scen. 3, 6: V 3×5×3 (45), Y 8×5×5 (200), 8 types



(d) Scen. 4: V 8×5×5 (200), Y 3×5×3 (45), 8 types



(e) Scen. 5, 7, 8: V/Y 8×5×5 (200 slots ea.), 8 types

Figure 6: *Visualizations of the five Basic SPGE experimental scenarios.*

Table 3: *Details of the Experimental Scenarios.* Sizes are Bays × Rows × Tiers.

Scenario	Vessel Sizes (Slots)	Yard Sizes (Slots)	Containers to Stow	Container Types	Cranes
1	3×5×3 (45)	3×5×3 (45)	45	3	1
2	3×5×3 (45)	3×5×3 (45)	45	8	1
3	3×5×3 (45)	8×5×5 (200)	45	8	1
4	8×5×5 (200)	3×5×3 (45)	45	8	1
5	8×5×5 (200)	8×5×5 (200)	200	8	1
6	3×5×3 (45)	8×5×5 (200)	45	8	3
7	8×5×5 (200)	8×5×5 (200)	200	8	3
8	8×5×5 (200)	8×5×5 (200)	200	8	5

5.5.2 Experimental Methodology

Before the formal training, we performed hyperparameter tuning for each scenario–algorithm pair. Specifically, we tuned the hyperparameters only in the Basic SPGE environment using Scenarios 1–5. For Scenarios 6, 7, and 8, each algorithm reused the hyperparameters from Scenarios 3, 5, and 5, respectively, as their problem settings are highly similar.

For tuning, we used the TPE sampler and Median pruner from the Optuna framework for automated optimization. For the relatively simple Scenarios 1 and 4, we ran 50 trials per algorithm, with each trial training for 10,000 timesteps (i.e., interactions between the agent and environment). The hyperparameters corresponding to the trial that performed best under deterministic evaluation were selected. For Scenarios 2, 3, and 5, the number of trials was increased to 100, and each trial was trained for 15,000 timesteps. When an algorithm still performs poorly under a certain scenario after automatic hyperparameter tuning, we attempt to use the optimal hyperparameter configuration from another scenario instead.

Each algorithm was assigned the same training timestep budget, based on the complexity of the scenario. Evaluation was conducted every 200 timesteps (or 500 timesteps for more complex scenarios), with 10 repeated evaluations per checkpoint, and the average of the evaluation metrics was recorded. To reduce the impact of randomness, we repeated the training experiments multiple times. Specifically, we ran 10 repetitions per scenario–algorithm pair for the Basic SPGE and LME environments, and 30 repetitions for SPGE-MC and SPAEC.

5.5.3 Experimental Infrastructure

For experimental facilities, we utilize the Academic Leiden Interdisciplinary Cluster Environment (ALICE), the HPC cluster of Leiden University for training in the LME environment. For some experiments, a laptop equipped with an Nvidia RTX 4050 (6GB VRAM), 32GB RAM, and an i7-13700H processor is used.

6 Results

This section reports the experimental results under the scenarios described in Section 5.5.1. Each group of experiments for a certain scenario was repeated multiple times. In the graphs, the solid lines are the mean metrics of the repeated experiments, and the shaded area indicates the variation across the repetitions.

6.1 Basic SPGE

Figures 7 and 8 show the training curves and the number of shifters during evaluation for five algorithms under each scenario. The horizontal axis represents the total timesteps, i.e., the count of interactions the agent experiences with the environment. The vertical axes show the episode reward during training and the number of shifters caused in a complete episode during evaluation, respectively. Here, we define the reward function to be the negative value of the shifters' count. During training, evaluation is performed every 200 timesteps.

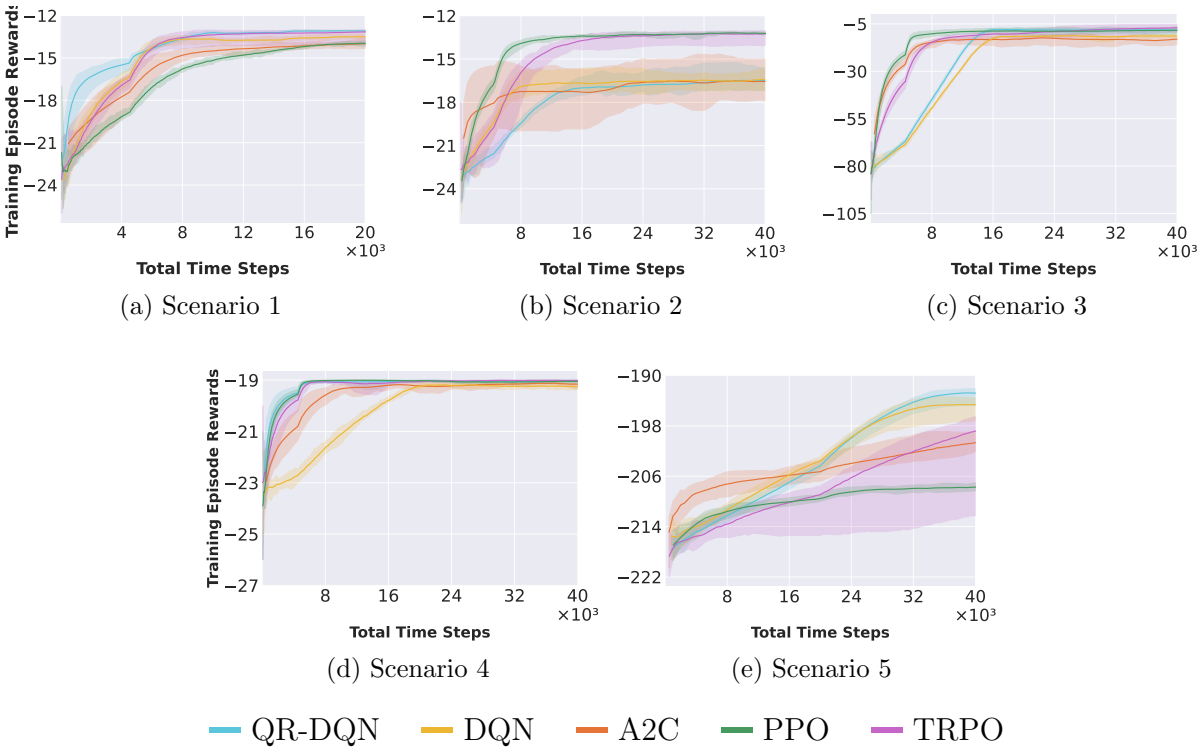
It can be seen that the agents generally perform better during evaluation than during training. This is because agents use exploratory policies during training, while deterministic policies are used in evaluation: for value-based methods, the agent selects the action with the highest Q-value; for actor-critic methods, the agent will take the most probable action instead of sampling from the distribution.

By examining the evaluation curves in Figure 7, it can be seen that overall, the A2C algorithm yields the worst-performing policy upon convergence during evaluation. Its performance is only comparable to other algorithms in Scenario 4, a relatively simple problem where long-term planning is not crucial.

The best-performing algorithm across all scenarios is TRPO. Even in the highly complex Scenario 5, it maintains a low total number of shifters toward the end of training (after 30,000 timesteps), stabilizing at around 193. PPO shows overall performance similar to TRPO, although it performs slightly worse in Scenario 5.

As for the value-based methods DQN and QR-DQN, they perform comparably to PPO and TRPO in simpler scenarios like Scenario 1, 3, and 4, and even in the complex Scenario 5, they still perform relatively well. Among the two, QR-DQN performs better than DQN.

However, as shown in Figures 7a and 7b, when facing a small-scale problem with 45 vessel slots and 45 yard slots, the increased complexity brought by a higher number of container types negatively impacts the training performance of DQN and QR-DQN compared to TRPO and PPO. This is further confirmed by the evaluation curves in Figures 8a and 8b: for the problem involving 8 container types, DQN, QR-DQN, and A2C all converge to policies that underperform relative to TRPO and PPO.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

(d) Scenario 4 (e) Scenario 5

Figure 7: *Training curves (episode reward vs. total timesteps) for five algorithms across different scenarios*. Subfigures (a)-(e) correspond to Scenarios 1-5 respectively. Higher reward values indicate better performance. Curves represent averages over 10 runs.

6.2 LME

Figure 9 shows the training curves of the agents corresponding to each algorithm in the LME environment for Scenario A, B, C described in section 5.5.1, while Figure 10 presents the mean shifter curves during evaluation.

As shown in Figure 9, all algorithms were able to converge successfully. In terms of evaluation, the A2C algorithm still performs relatively poorly on larger-scale problems, as shown in Figures 10b and 10c. From Figure 10c, we can see that TRPO and DQN slightly outperform QR-DQN and PPO during evaluation, although the differences in performance are minimal.

6.3 Single Agent vs. Multi-Agent

6.3.1 SPGE-MC

We use scenario 6, 7, 8 for experiments on SPGE-MC. For Scenario 6, evaluation was performed every 200 training timesteps. For Scenarios 7 and 8, evaluation was conducted every 500 training timesteps.

Figure 11 and 12 presents the curves for average episode operation time and average episode total shifters under these scenarios.

Both shifters and operation duration got reduced effectively. Among them, A2C still performed worse than the other algorithms in reducing shifters. Although DQN performed well in the

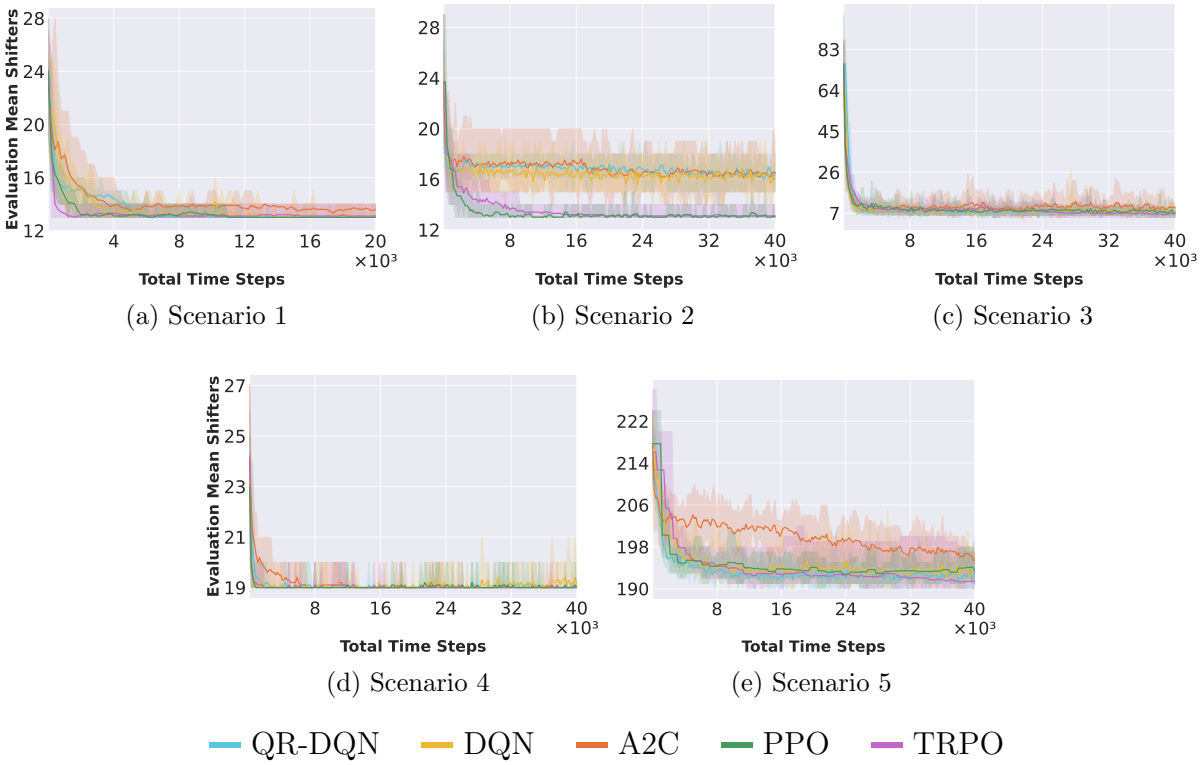


Figure 8: *Evaluation performance (number of shifters vs. total timesteps) for five algorithms across different scenarios.* Subfigures (a)-(e) correspond to Scenarios 1-5. Lower shifter counts indicate better performance. Data based on 10 evaluation trials and 10 training repetitions.

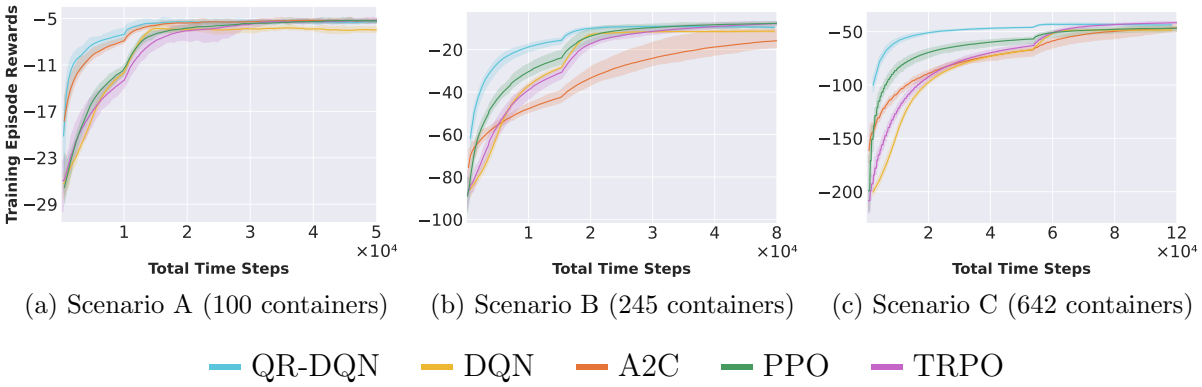
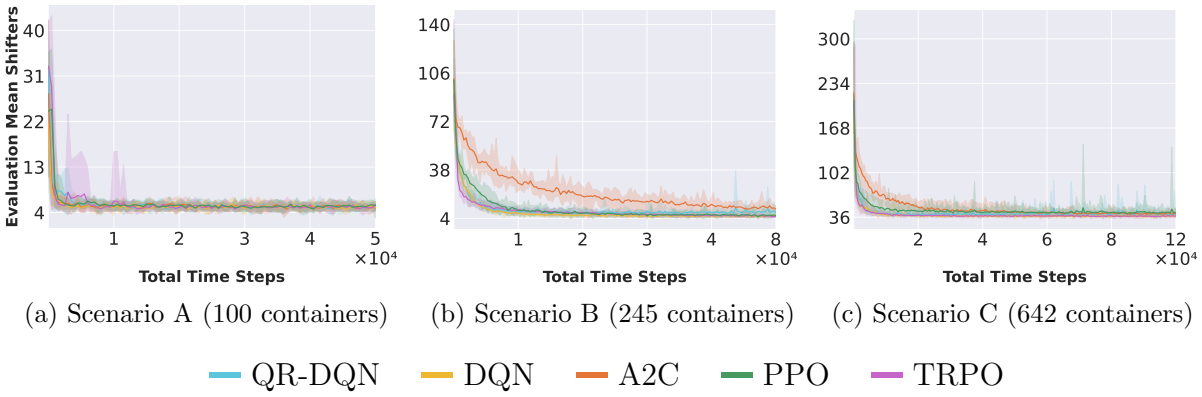
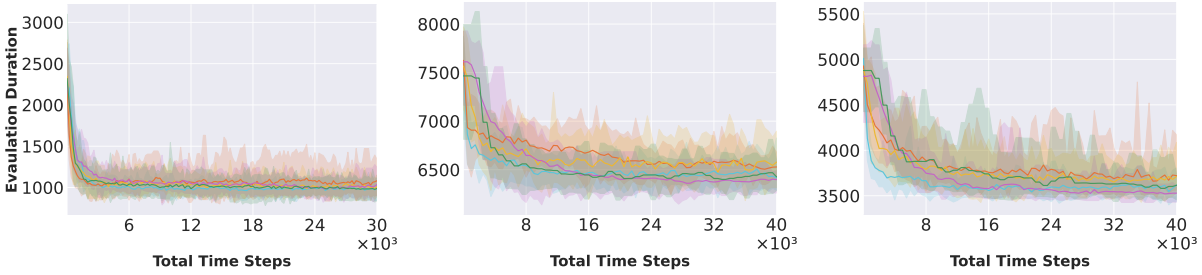


Figure 9: *Training curves (episode reward vs. total timesteps) for various algorithms in the Loadmaster.ai (LME) environment.* Subfigures (a), (b), and (c) correspond to scenarios with 100, 245, and 642 containers to be loaded, respectively. The reward function is based on the number of shifters (higher reward typically indicates fewer shifters), and the observation space includes only shifter information.



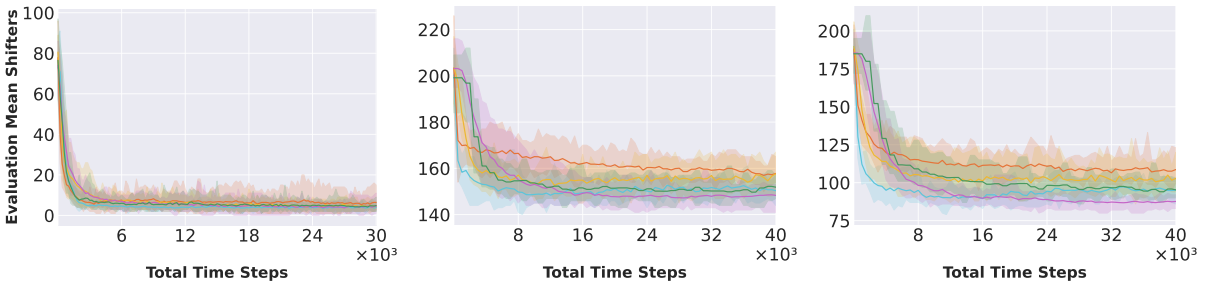
(a) Scenario A (100 containers) (b) Scenario B (245 containers) (c) Scenario C (642 containers)

Figure 10: *Evaluation performance (mean number of shifters vs. total timesteps) for various algorithms in the Loadmaster.ai (LME) environment.* Subfigures (a), (b), and (c) correspond to scenarios with 100, 245, and 654 containers. Evaluations were performed every 500 training timesteps, averaging over 10 episodes. Lower shifter counts indicate better performance.



(a) Scenario 6 (45 cont., 3 cranes) (b) Scenario 7 (200 cont., 3 cranes) (c) Scenario 8 (200 cont., 5 cranes)

Figure 11: *Average episode operation time (vs. total timesteps) for various algorithms in the SPGE-MC environment.* (a) Scenario 1: 45 containers, 3 cranes (eval. every 200 timesteps). (b) Scenario 2: 200 containers, 3 cranes (eval. every 500 timesteps, 30 training runs). (c) Scenario 3: 200 containers, 5 cranes (eval. every 500 timesteps, 30 training runs). Lower operation time indicates better performance.



(a) Scenario 6 (45 cont., 3 cranes) (b) Scenario 7 (200 cont., 3 cranes) (c) Scenario 8 (200 cont., 5 cranes)

QR-DQN DQN A2C PPO TRPO

Figure 12: *Average episode total shifters (vs. total timesteps) for various algorithms in the SPGE-MC environment.* (a) Scenario 6: 45 containers, 3 cranes (eval. every 200 timesteps). (b) Scenario 7: 200 containers, 3 cranes (eval. every 500 timesteps, 30 training runs). (c) Scenario 8: 200 containers, 5 cranes (eval. every 500 timesteps, 30 training runs). Lower shifter counts indicate better performance.

small-scale Scenario 6, its performance declined in the scenario with 200 containers, and its effectiveness in reducing operation time was comparable to that of A2C.

QR-DQN and PPO showed similar performance in reducing both shifters and operation time, outperforming A2C and DQN. TRPO performed the best overall, especially in the complex scenario with 200 containers, where it achieved the greatest reduction in shifters, surpassing the other algorithms.

6.3.2 SPAEC

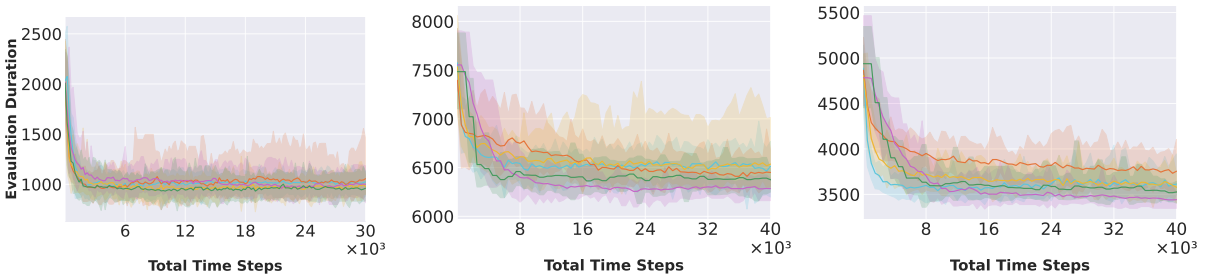
For SPAEC, the same three scenarios (6, 7, 8) and training/evaluation methods as described in Section 6.3.1 were used. Figure 13 and Figure 14 respectively show the mean operation time curve and the mean shifters curve during evaluation in the SPAEC environment.

We can observe algorithm performance patterns similar to those in Section 6.3.1: TRPO still performs generally the best in terms of complex scenarios (7 and 8), while A2C shows relatively poor performance, especially on shifter reducing. A few differences are worth noting: (1)For Scenario 7, in terms of reducing operation time, DQN performs worse than A2C. (2) In the scenario with 45 containers, QR-DQN performs best for shift reducing, while PPO performs best at minimizing operation time.

6.3.3 Evaluation Comparison

We summarize the evaluation KPIs of each algorithm after multiple training sessions in scenarios 6, 7, and 8 under SPGE-MC and SPAEC environments in Table 4 to explore the impact of single-agent versus multi-agent control approaches on agent performance. The evaluation data used for the summary are shown in Tables 10 and 11 in the appendix.

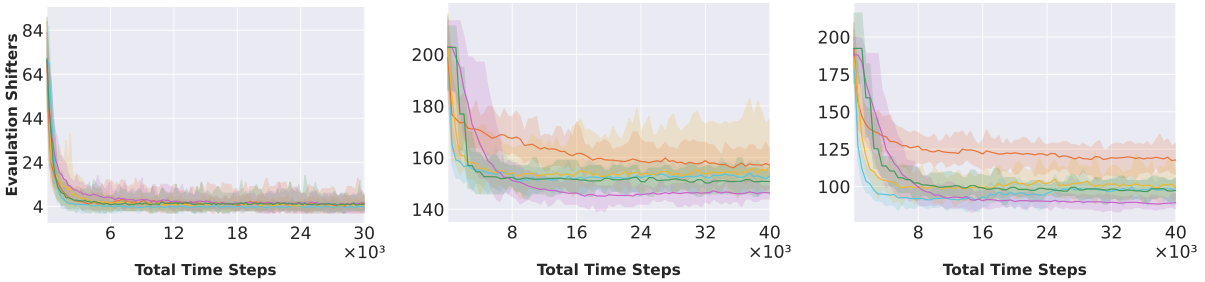
Table 4 contains the KPI mean values, standard deviations, and KPI differences caused by environmental variations for each scenario-algorithm pair. For the KPI differences, we employed Student's t-test and highlighted statistically significant differences ($p < 0.05$) with colors in



(a) Scenario 6 (45 cont., 3 cranes) (b) Scenario 7 (200 cont., 3 cranes) (c) Scenario 8 (200 cont., 5 cranes)

QR-DQN DQN A2C PPO TRPO

Figure 13: *Average episode operation time (vs. total timesteps) for various algorithms in the SPAEC environment.* (a) Scenario 1: 45 containers, 3 cranes (eval. every 200 timesteps). (b) Scenario 2: 200 containers, 3 cranes (eval. every 500 timesteps). (c) Scenario 3: 200 containers, 5 cranes (eval. every 500 timesteps). Lower operation time indicates better performance.



(a) Scenario 6 (45 cont., 3 cranes) (b) Scenario 7 (200 cont., 3 cranes) (c) Scenario 8 (200 cont., 5 cranes)

QR-DQN DQN A2C PPO TRPO

Figure 14: *Average episode total shifters (vs. total timesteps) for various algorithms in the SPGE-MC environment.* (a) Scenario 1: 45 containers, 3 cranes (eval. every 200 timesteps). (b) Scenario 2: 200 containers, 3 cranes (eval. every 500 timesteps, 30 training runs). (c) Scenario 3: 200 containers, 5 cranes (eval. every 500 timesteps, 30 training runs). Lower shifter counts indicate better performance.

Table 4: *Comparison of Final Evaluation Metrics (Shifters and Operation Time (s)) across SPGE-MC and SPAEC Environments.* Lower values are better for both KPIs. Values in parentheses indicate standard deviation. Best value per scenario/KPI/environment in **green**, worst in red. Difference (SMC-SPAEC): Blue if SPGE-MC is significantly better (negative diff), Purple if SPAEC is significantly better (positive diff). Only differences that are statistically significant based on t-tests are color-coded.

Algorithm	Scenario	KPI	SPGE-MC Value	SPAEC Value	Diff. (SMC-SPAEC)
TRPO	Scenario 6	Shifters	4.2 (2.25)	4.8 (2.82)	-0.5
		Op. Time	1025.9 (85.95)	983.8 (86.10)	42.1
	Scenario 7	Shifters	**148.2 (2.82)**	**146.3 (1.86)**	1.9
		Op. Time	**6401.2 (52.76)**	**6283.7 (61.26)**	117.6
	Scenario 8	Shifters	**87.7 (2.54)**	**89.3 (2.20)**	-1.6
		Op. Time	**3519.9 (47.36)**	**3431.0 (49.52)**	88.9
PPO	Scenario 6	Shifters	4.9 (1.84)	5.0 (1.85)	-0.1
		Op. Time	1003.8 (75.47)	**949.9 (85.14)**	54.0
	Scenario 7	Shifters	151.0 (3.07)	150.7 (2.80)	0.3
		Op. Time	6405.7 (88.58)	6380.6 (83.50)	25.2
	Scenario 8	Shifters	95.5 (3.73)	97.9 (3.52)	-2.4
		Op. Time	3616.5 (80.31)	3532.6 (70.03)	83.9
A2C	Scenario 6	Shifters	6.4 (3.85)	5.3 (2.50)	1.0
		Op. Time	1084.7 (143.37)	1049.0 (123.74)	35.7
	Scenario 7	Shifters	157.4 (3.09)	157.1 (3.22)	0.3
		Op. Time	6518.2 (84.91)	6451.8 (86.30)	66.4
	Scenario 8	Shifters	107.9 (6.03)	117.3 (6.32)	-9.3
		Op. Time	3699.7 (123.27)	3753.2 (162.01)	-53.5
DQN	Scenario 6	Shifters	4.4 (1.47)	4.3 (1.60)	0.1
		Op. Time	1000.6 (61.30)	982.6 (62.53)	18.0
	Scenario 7	Shifters	156.7 (5.06)	157.6 (8.08)	-0.9
		Op. Time	6577.9 (153.82)	6592.8 (260.21)	-14.9
	Scenario 8	Shifters	104.4 (6.99)	100.9 (6.97)	3.5
		Op. Time	3702.6 (131.09)	3603.4 (83.71)	99.3
QR-DQN	Scenario 6	Shifters	**3.9 (1.24)**	**3.5 (1.14)**	0.4
		Op. Time	**983.8 (60.57)**	1001.0 (74.36)	-17.2
	Scenario 7	Shifters	150.4 (3.72)	152.3 (2.79)	-1.8
		Op. Time	6412.4 (114.80)	6492.9 (155.58)	-80.5
	Scenario 8	Shifters	94.8 (4.11)	97.9 (4.11)	-3.1
		Op. Time	3607.2 (98.07)	3628.6 (126.93)	-21.3

the Diff. column.

Consistent with our observations in sections 6.3.1 and 6.3.2, in the simple scenario (Scenario 6), QR-DQN demonstrates better mean metric values; for complex scenarios (Scenarios 7 and 8), TRPO shows superior mean values for both KPIs. A2C performs worst in reducing shifters, while DQN shows poor performance in reducing operation time when facing complex scenarios.

The impact of single-agent versus multi-agent environments on agents exhibits high variability depending on the algorithm and scenario. Significant metric differences brought about by different formulations are more pronounced in complex scenarios, while for simple scenarios (Scenario 6), these differences are mostly non-significant. For complex scenarios with 200 containers, the single-agent control approach in the SPGE-MC environment generally performs significantly better in reducing shifters (with the exception of TRPO under Scenario 7); for significantly reducing operation time, the multi-agent approach corresponding to SPAEC shows better performance (such as TRPO, PPO, and A2C algorithms under Scenario 7, and TRPO, PPO, and DQN algorithms under Scenario 8).

7 Discussion

From the experiments above, we observe that the performance of different reinforcement learning algorithms on solving the CSPP is highly dependent on the complexity of the problem, the optimization objectives, and the environment formulation.

For simpler scenarios and problem settings with a single optimization objective (as in Scenario 1 for the Basic SPGE), or when the representation of environment information is simplified (as in LME), DQN, QR-DQN, PPO, A2C, and TRPO exhibit similar performance and are effective in reducing shifters. However, as the problem becomes more complex, the performance of A2C shows a noticeable decline. This may be related to the accuracy of the critic network in estimating state values: in training A2C, we adopted the original paper's choice of 5 for the n-step parameter, but in complex problems, the true value of a single action may not be reflected solely by the immediate shifter, whose long-term impact might only emerge far beyond 5 steps. This may lead to significant bias in the advantage estimation based on 5-step returns. Furthermore, unlike PPO and TRPO, A2C does not use Generalized Advantage Estimation (GAE), which allows for a trade-off between bias and variance in value estimation, and thus cannot benefit from the improved credit assignment that GAE provides.

When examining Scenario 2 of the Basic SPGE, we observe that value-based methods also suffer from sub-optimality in more complex settings. Compared to scenarios with a larger action space (e.g., those with 8 container types), Scenario 2, despite having a smaller action space, contains more local optima traps due to the interleaved slot types on the vessel. These traps represent seemingly feasible short-term decisions that hinder future optimization. DQN and QR-DQN may fail to learn Q-values that capture such subtle distinctions. In contrast, TRPO and PPO aim for monotonic policy improvement by constraining the magnitude of policy updates, giving them an advantage over DQN and QR-DQN in avoiding sub-optimal solutions.

In relatively simple scenarios, value-based methods, especially QR-DQN, perform best (Scenario 6). This may be due the Q-function of the state (or action) is easy for the Q network to learn, and QR-DQN's rich representation of Q value information can help it converge to a

high-quality solution faster.

In very complicated scenarios (Scenario 5 for Basic SPGE, Scenarios 7 and 8 for SPGE-MC and SPAEC), TRPO shows the best performance. This may be due to TRPO solving a constrained optimization problem at each iteration, which enforces stricter improvements in the policy.

For the two different formulations, single-agent and multi-agent, their impact becomes significant only in more complex scenarios (Scenario 7 and 8) and with certain algorithms. This impact cannot be generalized as one formulation being universally better. A trend we can observe is that TRPO is the algorithm most likely to exhibit notable performance differences between the two formulations in terms of both reducing shifters and operation time under distinct scenario-algorithm combinations. Moreover, the single-agent formulation tends to be more advantageous for optimizing shifters in most cases.

As discussed in Section 5.2, due to SPAEC's agent selection mechanism, the single-agent setup in SPGE-MC has more flexibility in crane selection when multiple cranes are available at a given step, which might be helpful for agent to learn a globally optimal policy for minimizing total shifters. While minimizing shifters and minimizing total operation time can become conflicting objectives beyond a certain optimization threshold, which requiring trade-offs, it might be easier for agent to prioritize shifter reducing since shifter-based rewards are denser.

8 Conclusion

This thesis aims to explore the effectiveness of using reinforcement learning methods to solve the container stowage planning problem (CSPP), examining differences based on algorithms and problem scales, as well as the impact of single-agent versus multi-agent formulations on RL algorithm performance when combining crane scheduling aspects with CSPP. When crane scheduling is not considered, the objective is shifter minimization. Once crane scheduling is included, the objective is extended to also minimize total operation time. We selected value-based algorithms DQN and QR-DQN, as well as actor-critic architecture algorithms A2C, TRPO, and PPO for experiments. The results show that all these algorithms can all achieve promising outcomes in solving CSPP and optimizing the aforementioned objectives, although their performance varies.

We proposed three research sub-questions in the Introduction section to explore the performance differences, and address them through a series of comparative experiments across multiple designed scenarios.

1. *How do different RL algorithms perform on the container stowage planning problem in the same environment?*

   This depends on the problem scale and the representation of environment information. We observe that when the environment representation is simple or the problem is relatively easy, the five aforementioned algorithms are able to achieve similarly good performance in reducing the target metrics we care about.

2. *How does problem scale affect the performance of RL algorithms?*

   As the problem complexity increases, the A2C algorithm shows a clear decline in performance, particularly in reducing the number of shifters. The DQN algorithm also ex-

periences a similar issue, though it is more evident in its degradation in operation time minimization. Moreover, when the scenario contains many suboptimal solution traps, A2C, DQN, and QR-DQN are more likely to fall into these suboptimal solutions. The impact of increasing problem complexity is relatively minimal for PPO and TRPO compared to the other algorithms, while TRPO shows the best performance in the most complex scenarios.

3. *When crane scheduling is considered together with CSPP, how do single-agent vs. multi-agent formulations influence algorithm performance?*

   The impact of these two formulations on algorithm performance is complex and heterogeneous across environments and algorithms. In scenarios with simpler problem scales, the difference between single-agent and multi-agent formulations is not significant. In more complex settings, they are more likely to produce noticeable effects, though not necessarily in a consistent or directional manner. What we do observe is that, overall, the single-agent formulation tends to perform better in reducing shifters during algorithm training.

In summary, this study highlights the complex performance variations that arise when applying RL methods to solve the container stowage planning problem, depending on the choice of optimization objective, problem scale, and problem formulation. Among the evaluated methods across experiments, TRPO is the most promising algorithm in complex scenarios. However, in practical applications, the most suitable approach should be chosen based on the specific needs of the problem.

Future Work

This thesis has covered a comprehensive comparison of various RL algorithms across different CSPP scenarios, as well as the impact of single-agent versus multi-agent formulations. However, there remains ample room for future exploration. Future work can focus on richer scenario settings, simulation fidelity of the SPGE environment, and consideration of additional aspects of CSPP.

For scenario design, future research could include more complex setups to further test the potential of different algorithms, such as scenarios involving the stowage of more than 1,000 containers. Moreover, the scenarios in this study only involve 20-foot containers, each occupying a single slot. Designing scenarios including 40-foot containers can increase complexity and better reflect real-world operations.

Another area that can be improved is the simulation fidelity of SPGE. The SPGE in this study has made certain simplifications, such as abstracting the vessel structure into a cubic form, omitting the implementation of hatches, and modeling container stowage time using container-specific random values. Future research can enhance the realism by implementing ship hulls that better reflect actual vessel structures, adding hatches and certain zones (e.g., power grids), as well as adopting more sophisticated modeling for operation time.

Additionally, other complex aspects of CSPP can also be considered. This study only includes two optimization objectives: minimizing shifter and operation time. Future scenarios could incorporate containers' port of discharge information to optimize for re-stowage minimization during later port calls, or address the challenge of weight distribution on the vessel. Explor-

ing how different RL algorithms manage trade-offs between potentially conflicting objectives presents a valuable direction for further investigation.

References

Ambrosino, D., Anghinolfi, D., Paolucci, M., and Sciomachen, A. (2010). An experimental comparison of different heuristics for the master bay plan problem. In *Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings 9*, pages 314–325. Springer.

Ambrosino, D., Sciomachen, A., and Tanfani, E. (2004). Stowing a containership: the master bay plan problem. *Transportation Research Part A: Policy and Practice*, 38(2):81–99.

Avriel, M. and Penn, M. (1993). Exact and approximate solutions of the container ship stowage problem. *Computers & industrial engineering*, 25(1-4):271–274.

Azevedo, A. T., de Salles Neto, L. L., Chaves, A. A., and Moretti, A. C. (2018). Solving the 3d stowage planning problem integrated with the quay crane scheduling problem by representation by rules and genetic algorithm. *Applied Soft Computing*, 65:495–516.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR.

Bellman, R. (1966). Dynamic programming. *science*, 153(3731):34–37.

Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.

Bono, S., Madan, S., Grover, I., Yasueda, M., Breazeal, C., Pfister, H., and Kreiman, G. (2025). The indoor-training effect: unexpected gains from distribution shifts in the transition function.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Cho, J. and Ku, N. (2024). Developing a container ship loading-planning program using reinforcement learning. *Journal of Marine Science and Engineering*, 12(10):1832.

Cruz-Reyes, L., Hernandez Hernandez, P., Melin, P., Mar-Ortiz, J., Fraire Huacuja, H. J., Puga Soberanes, H. J., and Gonzalez Barbosa, J. J. (2015). Lower and upper bounds for the master bay planning problem. *International Journal of Combinatorial Optimization Problems and Informatics*, 6(1):42–52.

Dabney, W., Rowland, M., Bellemare, M., and Munos, R. (2018). Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Garratt, M. G. (1980). Ro-ro versus lo-lo: a matter of scale. *Maritime Policy & Management*, 7(4):223–232.

He, J., Zhang, L., Deng, Y., Yu, H., Huang, M., and Tan, C. (2023). An allocation approach for external truck tasks appointment in automated container terminal. *Advanced Engineering Informatics*, 55:101864.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Hsu, H.-P., Wang, C.-N., Fu, H.-P., and Dang, T.-T. (2021). Joint scheduling of yard crane, yard truck, and quay crane for container terminal considering vessel stowage plan: An integrated simulation-based optimization approach. *Mathematics*, 9(18):2236.

Hu, W., Hu, Z., Shi, L., Luo, P., and Song, W. (2012). Combinatorial optimization and strategy for ship stowage and loading schedule of container terminal. *J. Comput.*, 7(8):2078–2092.

Huang, S. and Ontañón, S. (2020). A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*.

Huang, S., Ontañón, S., Bamford, C., and Grela, L. (2021). Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE.

Jayawardana, V., Tang, C., Li, S., Suo, D., and Wu, C. (2022). The impact of task underspecification in evaluating deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:23881–23893.

Jensen, R. M., Pacino, D., Ajspur, M. L., and Vesterdal, C. (2018). *Container vessel stowage planning*. Weilbach.

Jiang, T., Zeng, B., Wang, Y., and Yan, W. (2021). A new heuristic reinforcement learning for container relocation problem. In *Journal of Physics: Conference Series*, volume 1873, page 012050. IOP Publishing.

Kizilay, D. and Eliiyi, D. T. (2021). A comprehensive review of quay crane scheduling, yard operations and integrations thereof in container terminals. *Flexible Services and Manufacturing Journal*, 33(1):1–42.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Li, F., Tian, C., Cao, R., and Ding, W. (2008). An integer linear programming for container stowage problem. In *Computational Science–ICCS 2008: 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part I 8*, pages 853–862. Springer.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PmLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Pacino, D., Delgado, A., Jensen, R. M., and Bebbington, T. (2011). Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. In *International conference on computational logistics*, pages 286–301. Springer.

Padakandla, S. (2021). A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys (CSUR)*, 54(6):1–25.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Reda, D., Tao, T., and van de Panne, M. (2020). Learning to locomote: Understanding how environment design matters for deep reinforcement learning. In *Motion, Interaction and Games*, MIG '20. ACM.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sciomachen, A. and Tanfani, E. (2003). The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem. *IMA Journal of Management Mathematics*, 14(3):251–269.

Shen, Y., Zhao, N., Xia, M., and Du, X. (2017). A deep q-learning network for ship stowage planning problem. *Polish Maritime Research*, 24(s3):102–109.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., et al. (2021). Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043.

Trade, U. and Development (2024). *Review of Maritime Transport 2024*. United Nations, 2024 edition.

van Twiller, J., Grbic, D., and Jensen, R. M. (2023). Towards a deep reinforcement learning model of master bay stowage planning. In *International Conference on Computational Logistics*, pages 105–121. Springer.

van Twiller, J., Sivertsen, A., Pacino, D., and Jensen, R. M. (2024). Literature survey on the container stowage planning problem. *European Journal of Operational Research*, 317(3):841–857.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

Wei, L., Wei, F., Schmitz, S., and Kunal, K. (2021). Optimization of container relocation problem via reinforcement learning. *Logistics Journal: Proceedings*, 2021(17).

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Wilson, I. D. and Roach, P. A. (2000). Container stowage planning: a methodology for generating computerised solutions. *Journal of the Operational Research Society*, 51(11):1248–1255.

Wolgast, T. and Nieße, A. (2024). Learning the optimal power flow: Environment design matters. *Energy and AI*, 17:100410.

Xia, M., Li, Y., Shen, Y., and Zhao, N. (2020). Loading sequencing problem in container terminal with deep q-learning. *Journal of Coastal Research*, 103(SI):817–821.

Zhao, N., Guo, Y., Xiang, T., Xia, M., Shen, Y., and Mi, C. (2018). Container ship stowage based on monte carlo tree search. *Journal of Coastal Research*, (83):540–547.

Zhao, W., Queralta, J. P., and Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE.

Zheng, K., Lu, Z., and Sun, X. (2010). An effective heuristic for the integrated scheduling problem of automated container handling system using twin 40'cranes. In *2010 second international conference on computer modeling and simulation*, volume 1, pages 406–410. IEEE.

Zhou, C., Zhu, S., Bell, M. G., Lee, L. H., and Chew, E. P. (2022). Emerging technology and management research in the container terminals: Trends and the covid-19 pandemic impacts. *Ocean & Coastal Management*, 230:106318.

Zhu, H., Ji, M., and Guo, W. (2020). Integer linear programming models for the containership stowage problem. *Mathematical Problems in Engineering*, 2020(1):4382745.

A Hyper-parameters

This section contains the hyperparameter values used in the experiments of this thesis. All unspecified hyperparameters use the default values from the algorithms provided in Stable-Baselines3.

Regarding the network architecture, for all algorithms, the input to the neural network is the same as the environment's observation space, and the output corresponds to the environment's action space (except for QR-DQN, whose output is the action space size multiplied by the number of quantiles. In all our experiments, the number of quantiles is set to the default value of 200 in SB3). All networks use hidden layers with 64 neurons.

All algorithms use the same hyperparameters as in Scenario 3 for Scenario 6, and the same hyperparameters as in Scenario 5 for Scenarios 7 and 8.

Table 5: Hyperparameters for TRPO Algorithm across 8 Scenarios.

Hyperparameter	**Sc. 1**	**Sc. 2**	**Sc. 3**	**Sc. 4**	**Sc. 5**	**Sc. A**	**Sc. B**	**Sc. C**
Learning Rate	2e-4	2e-3	3e-4	3e-4	5e-3	1e-3	6e-4	6e-4
Gamma	0.26	0.88	0.72	0.72	0.53	0.30	0.33	0.57
GAE Lambda	0.81	0.89	0.96	0.96	0.93	0.85	0.84	0.88
N-steps	64	128	32	32	512	256	256	256
Hidden Layers	2	2	1	1	2	2	1	1
Activation Function	ReLU	Tanh	ReLU	ReLU	ReLU	Tanh	ReLU	Tanh
Target KL	8e-3	8e-3	9e-3	9e-3	5e-3	1e-3	6e-3	8e-3
CG Damping	0.02	0.02	0.04	0.04	0.01	8e-3	2e-3	4e-3

Table 6: Hyperparameters for PPO Algorithm across 8 Scenarios.

Hyperparameter	**Sc. 1**	**Sc. 2**	**Sc. 3**	**Sc. 4**	**Sc. 5**	**Sc. A**	**Sc. B**	**Sc. C**
Learning Rate	5e-4	1e-3	4e-5	4e-5	3e-5	2e-4	2e-3	2e-3
Gamma	0.67	0.94	0.31	0.31	0.20	0.30	0.56	0.56
GAE Lambda	0.87	0.91	0.84	0.84	0.98	0.92	0.98	0.98
N-steps	32	256	64	64	1024	512	256	256
Hidden Layers	1	2	1	1	1	1	1	1
Activation Function	Tanh	Tanh	ReLU	ReLU	ReLU	Tanh	Tanh	Tanh
Entropy Coef	3e-3	7e-4	6e-5	6e-5	3e-3	0	3e-6	3e-6
Max Grad. Norm.	1.11	0.83	4.38	4.38	4.94	0.5	3.32	3.32

Table 7: Hyperparameters for A2C Algorithm across 8 Scenarios.

Hyperparameter	**Sc. 1**	**Sc. 2**	**Sc. 3**	**Sc. 4**	**Sc. 5**	**Sc. A**	**Sc. B**	**Sc. C**
Gamma	0.99	0.40	0.42	0.99	0.58	0.46	0.30	0.81
Hidden Layers	1	2	2	1	1	2	2	1
Activation Function	Tanh	ReLU	Tanh	Tanh	Tanh	Tanh	Tanh	Tanh

Table 8: Hyperparameters for DQN Algorithm across 8 Scenarios.

Hyperparameter	**Sc. 1**	**Sc. 2**	**Sc. 3**	**Sc. 4**	**Sc. 5**	**Sc. A**	**Sc. B**	**Sc. C**
Learning Rate	1e-4	4e-4	8e-3	0.03	5e-4	8e-4	7e-4	7e-4
Gamma	0.99	0.65	0.48	0.55	0.25	0.21	0.40	0.40
Learning Starts	100	92	105	95	157	41	161	161
Exploration Fraction	0.1	0.2	0.32	0.43	0.40	0.1	0.1	0.1
Tau	1.0	2e-3	1.0	0.02	1e-3	0.03	4e-3	4e-3
Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	Tanh	Tanh
Target Update Interval	1e4	—	2649	—	—	—	—	—

Table 9: Hyperparameters for QR-DQN Algorithm across 8 Scenarios.

Hyperparameter	**Sc. 1**	**Sc. 2**	**Sc. 3**	**Sc. 4**	**Sc. 5**	**Sc. A**	**Sc. B**	**Sc. C**
Learning Rate	5e-5	1e-4	9e-5	5e-5	5e-4	5e-4	2e-3	2e-3
Gamma	0.99	0.33	0.36	0.99	0.60	0.99	0.30	0.30
Learning Starts	100	81	158	100	48	188	111	111
Exploration Fraction	5e-3	0.29	0.29	5e-3	0.47	5e-3	5e-3	5e-3
Tau	1.0	1.0	0.008	1.0	1.0	0.04	1.0	1.0
Activation Function	ReLU	Tanh	Tanh	ReLU	ReLU	ReLU	Tanh	Tanh
Target Update Interval	1e4	4776	—	1e4	8112	—	172	172

B Detailed Evaluation Data

Table 10: Raw Shifter Values (Mean of 10 Evaluations per Training Run) for Each Algorithm, Scenario, and Environment. N=30 runs per combination.

Algo.	Scen.	Env.	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30
TRPO	Sc. 6	MC	920	1049	1074	1010	996	1003	1027	1095	1051	1204	971	1030	992	915	1074	979	1245	1019	1081	931	1076	1020	1116	975	1013	982	970	801	1027	1132
		AEC	998	1027	1089	1030	942	815	925	1162	1184	997	865	970	990	964	947	1018	993	1125	955	1076	932	883	966	1054	918	933	1039	934	888	896
	Sc. 7	MC	6320	6359	6349	6260	6439	6340	6387	6416	6489	6409	6370	6466	6417	6465	6395	6446	6466	6445	6333	6375	6470	6373	6390	6453	6378	6415	6446	6423	6374	6369
		AEC	6254	6194	6299	6340	6387	6338	6280	6266	6270	6326	6275	6347	6181	6298	6161	6341	6266	6297	6274	6303	6291	6405	6358	6260	6190	6244	6240	6335	6183	6307
	Sc. 8	MC	3516	3601	3606	3516	3575	3529	3584	3481	3430	3496	3484	3511	3480	3503	3607	3517	3490	3510	3537	3596	3575	3495	3510	3531	3535	3503	3476	3456	3496	3451
		AEC	3480	3405	3351	3493	3441	3440	3504	3427	3379	3523	3415	3475	3424	3497	3424	3501	3378	3347	3461	3468	3425	3313	3428	3410	3419	3424	3391	3434	3450	3402
PPO	Sc. 6	MC	1114	1014	962	1025	1009	1037	972	984	1058	925	850	1008	982	994	949	838	1072	900	1029	1033	949	1120	1008	1026	1145	1160	1046	954	965	987
		AEC	979	1082	1022	886	1062	978	1021	1114	919	897	936	839	871	977	942	832	909	900	1129	924	936	795	914	897	901	963	1094	1006	876	895
	Sc. 7	MC	6374	6471	6380	6595	6335	6503	6376	6557	6398	6476	6450	6284	6500	6472	6433	6322	6412	6288	6439	6467	6336	6444	6340	6224	6352	6399	6306	6356	6552	6331
		AEC	6415	6443	6250	6379	6421	6366	6297	6484	6274	6412	6446	6397	6257	6459	6401	6276	6376	6386	6481	6345	6344	6347	6558	6374	6438	6452	6372	6448	6152	6367
	Sc. 8	MC	3495	3628	3644	3570	3609	3666	3651	3653	3512	3604	3655	3559	3692	3586	3643	3662	3456	3710	3568	3741	3617	3494	3853	3645	3568	3573	3691	3617	3563	3570
		AEC	3447	3404	3472	3544	3600	3505	3608	3606	3594	3497	3516	3507	3644	3600	3513	3500	3522	3484	3443	3526	3475	3629	3555	3523	3631	3615	3594	3368	3491	3564
A2C	Sc. 6	MC	1176	1035	1101	1063	1346	1100	1073	923	1100	1315	997	989	907	1277	1137	984	955	1101	926	1015	1180	1521	1112	1006	1028	936	1178	1180	939	942
		AEC	929	933	978	1358	909	992	1164	1033	1006	1462	1088	949	961	1014	1131	1046	988	979	1092	1016	1098	1092	1037	1064	917	879	1051	1054	1178	1072
	Sc. 7	MC	6430	6589	6529	6409	6458	6626	6475	6461	6436	6566	6660	6681	6482	6518	6665	6434	6500	6572	6425	6644	6657	6558	6516	6424	6545	6446	6448	6438	6478	6475
		AEC	6360	6508	6330	6684	6374	6521	6345	6377	6457	6493	6576	6416	6454	6397	6416	6489	6480	6487	6296	6501	6469	6468	6411	6646	6434	6465	6336	6443	6482	6439
	Sc. 8	MC	3566	3600	3725	3931	3636	3734	3667	3769	3707	3701	3581	3793	3605	3756	3724	3694	4170	3727	3563	3775	3629	3633	3670	3726	3619	3619	3801	3700	3534	3637
		AEC	3521	3831	3678	3874	3701	3946	3682	3888	3554	3598	3913	3706	3555	4103	4029	3634	3700	3825	3699	3666	3888	3514	3718	3701	3832	3570	3759	3646	3758	4107
DQN	Sc. 6	MC	995	1004	916	990	955	1016	890	1054	1088	1048	1152	1073	966	1017	1068	944	1013	967	1012	983	885	947	1075	1003	969	953	1088	979	1012	956
		AEC	896	1023	1092	955	942	1032	951	977	920	1013	907	913	1084	982	1044	983	921	1016	938	920	1016	1087	943	1018	1048	936	882	949	1099	990
	Sc. 7	MC	6335	6638	6638	6613	6484	6615	6400	6429	6730	6744	6403	6562	6727	6746	6829	6402	6314	6480	6897	6588	6690	6488	6598	6688	6476	6423	6446	6743	6735	6475
		AEC	6405	6479	6404	6644	6500	6556	6517	6306	6376	6506	6535	6420	6479	6945	6525	7013	6503	6904	7132	6395	6904	6475	6263	6480	6472	6435	6387	7020	6548	7256
	Sc. 8	MC	3829	3668	3614	3777	3633	3534	3882	3567	3627	3692	3666	3737	3960	3545	3712	3647	3546	3824	4086	3685	3581	3840	3822	3572	3653	3741	3698	3752	3593	3596
		AEC	3581	3634	3588	3605	3505	3497	3618	3579	3408	3607	3439	3640	3620	3616	3571	3730	3678	3554	3540	3560	3720	3635	3728	3576	3662	3765	3611	3710	3505	3619
QR-DQN	Sc. 6	MC	1071	949	841	1145	989	1077	1003	962	892	958	989	953	1004	986	972	971	1050	967	980	984	962	949	1033	978	902	1063	1025	898	982	979
		AEC	1008	1060	1093	881	1033	960	1105	965	992	905	1016	1043	1122	999	1019	1057	1079	1068	947	1114	1015	939	1048	909	988	959	895	976	815	1021
	Sc. 7	MC	6447	6578	6462	6290	6827	6396	6347	6361	6349	6417	6360	6332	6506	6289	6458	6377	6339	6332	6373	6404	6485	6329	6428	6430	6276	6424	6360	6455	6297	6644
		AEC	6599	6633	6501	6551	6321	6334	6638	6605	6390	6665	6582	6514	6390	6561	6490	6768	6542	6198	6573	6424	6214	6348	6382	6343	6537	6347	6362	6840	6437	6697
	Sc. 8	MC	3622	3629	3805	3469	3723	3515	3610	3696	3733	3525	3517	3754	3567	3614	3657	3433	3533	3627	3728	3529	3540	3833	3563	3587	3620	3585	3524	3579	3505	3595
		AEC	3690	3673	3527	3565	3500	3524	3500	3499	3537	3699	3684	3945	3657	3615	3556	3792	3522	3514	3691	3586	3455	3429	3770	3639	3923	3721	3772	3605	3642	3625

Table 11: Raw Shifter Values (Mean of 10 Evaluations per Training Run) for Each Algorithm, Scenario, and Environment. N=30 runs per combination.

Algo.	Scen.	Env.	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30
TRPO	Sc. 6	MC	1	6	4	3	6	1	5	7	4	5	6	6	3	2	4	4	11	2	5	6	6	4	7	4	1	2	4	1	2	5
		AEC	7	8	9	5	4	6	5	9	8	9	1	9	3	3	4	3	5	4	3	5	3	2	3	11	2	1	4	5	1	1
	Sc. 7	MC	147	150	144	146	149	141	148	151	151	148	153	151	149	151	147	147	149	145	149	150	149	150	145	151	143	150	151	150	146	145
		AEC	144	145	145	149	147	147	146	146	144	145	149	145	146	148	147	149	146	147	145	145	146	150	148	145	142	144	148	146	146	149
	Sc. 8	MC	90	88	87	92	91	86	88	88	85	86	90	88	86	83	89	86	87	84	90	89	90	87	83	87	93	91	86	85	86	89
		AEC	95	90	88	91	94	88	88	86	89	89	89	89	85	89	87	91	88	91	91	88	89	86	93	88	90	88	90	89	90	90
PPO	Sc. 6	MC	6	5	2	5	4	4	5	5	4	3	3	5	6	7	6	3	9	3	7	5	4	9	8	3	4	7	5	4	3	3
		AEC	6	2	6	4	9	4	6	4	6	6	6	4	3	5	5	5	3	3	5	6	9	6	4	1	6	4	6	7	6	2
	Sc. 7	MC	150	157	147	156	151	152	154	157	150	154	150	147	156	153	152	148	150	148	152	153	152	149	151	147	147	147	153	148	150	150
		AEC	147	155	149	154	152	150	155	152	150	148	153	152	148	155	150	149	151	150	154	149	146	148	157	153	148	152	150	148	149	148
	Sc. 8	MC	96	90	98	91	94	92	94	91	94	93	92	94	96	100	103	98	96	98	96	97	98	90	105	100	92	91	97	96	99	95
		AEC	97	91	100	100	98	97	100	98	99	95	93	92	100	106	96	94	98	99	96	98	96	105	96	95	104	99	100	95	102	99
A2C	Sc. 6	MC	13	6	4	6	9	6	4	4	6	15	6	3	4	4	9	6	5	9	3	4	8	19	6	3	2	2	9	8	4	4
		AEC	4	2	6	9	4	5	5	3	1	9	7	5	6	9	4	8	6	1	4	6	10	5	4	4	5	3	6	11	4	4
	Sc. 7	MC	157	159	156	156	162	165	156	155	154	158	156	158	163	159	157	159	154	157	155	160	162	157	162	156	156	156	156	152	158	152
		AEC	154	158	152	160	154	163	152	154	161	162	160	155	157	155	156	159	155	160	156	159	157	157	160	162	160	154	152	153	157	159
	Sc. 8	MC	105	100	100	113	106	115	102	113	107	110	104	115	103	115	113	105	123	107	101	116	102	103	106	116	107	101	115	103	106	106
		AEC	108	122	113	123	118	125	114	120	108	110	122	117	108	129	125	110	120	129	119	115	119	108	116	117	126	112	119	110	115	121
DQN	Sc. 6	MC	3	7	3	5	2	4	2	6	4	6	6	7	5	5	4	5	3	6	6	5	1	3	4	4	4	4	4	5	5	3
		AEC	2	4	5	4	4	4	6	4	3	4	2	4	4	7	9	5	4	5	6	3	6	4	3	6	6	3	4	2	4	2
	Sc. 7	MC	156	164	154	153	158	163	154	152	162	167	151	154	159	164	158	149	155	158	164	156	158	155	151	158	159	153	145	162	157	152
		AEC	148	154	150	156	156	155	155	149	151	162	158	152	158	175	157	169	155	166	170	150	169	153	143	157	154	158	152	170	153	173
	Sc. 8	MC	103	111	110	100	107	96	110	97	109	96	102	99	100	99	99	99	94	109	120	97	96	115	116	105	104	112	102	111	101	112
		AEC	103	103	106	99	103	93	99	96	88	89	94	97	101	105	97	109	102	92	101	103	91	106	115	100	115	114	105	103	100	98
QR-DQN	Sc. 6	MC	5	5	2	6	5	4	6	3	2	3	5	3	5	4	4	3	3	4	4	5	3	4	2	2	4	4	6	4	2	5
		AEC	3	3	6	3	4	2	4	3	2	5	4	4	4	2	4	2	5	4	3	5	6	4	4	2	3	3	4	3	2	3
	Sc. 7	MC	157	153	149	148	157	148	152	156	146	147	144	151	151	147	147	154	147	150	150	150	153	145	150	152	155	151	149	146	150	158
		AEC	153	154	156	152	150	152	152	153	152	156	154	156	148	156	150	155	155	149	154	153	147	150	152	149	153	148	151	157	148	153
	Sc. 8	MC	97	99	100	86	97	95	97	92	96	97	97	92	102	91	95	91	93	94	96	91	101	101	94	96	90	97	94	95	84	94
		AEC	96	98	95	99	100	95	92	94	95	99	102	105	101	99	95	96	88	96	100	102	91	93	106	97	101	102	101	98	102	99