Autoregressive DRL for Multi-Robot Scheduling in Semiconductor Cluster Tools

Soo-Hwan Cho
School of Electrical Engineering
Korea University, Seoul, South Korea
Email: soohwancho@korea.ac.kr

Jean Seong Bjorn Choe
School of Electrical Engineering
Korea University, Seoul, South Korea
Email: garangg@korea.ac.kr

Jong-Kook Kim
School of Electrical Engineering
Korea University, Seoul, South Korea
Email: jongkook@korea.ac.kr

***Abstract*—Deep reinforcement learning (DRL) has been widely applied to multi-robot scheduling, including semiconductor cluster tools, where maximizing throughput is critical. These tools operate under strict constraints, requiring precise coordination for efficient operation. However, managing multiple robots in such complex environments remains challenging. We propose an autoregressive DRL framework that sequentially generates robot actions using dynamic action masking, enabling context-aware decision making in large discrete action spaces. The agent is guided by a reward function that promotes step-wise progress, task completion, and reduced travel distance, encouraging efficient use of resources. Our approach demonstrates strong performance across representative tool configurations, highlighting the value of policy decomposition in complex scheduling tasks.**

I. Introduction

Semiconductor cluster tools are critical manufacturing systems composed of multiple process modules (PMs) [1]–[3]. These tools are widely used in the front-end stage of semiconductor fabrication, particularly in photolithography, etching, and cleaning, where precise wafer transport is essential for maintaining throughput and yield [4], [5].

Cluster tools in semiconductor manufacturing are broadly classified into radial-type and track-type architectures [4]. Radial-type systems, with centralized wafer flow, are widely studied due to their structured operations and well-defined scheduling patterns [1], [6]–[8]. In contrast, track-type systems have more distributed paths, supporting a larger number of robots and more flexible wafer routing, but also present more diverse and less standardized scheduling challenges. This work addresses these challenges by developing scalable DRL methods applicable to both radial and track-type environments.

To address these challenges, we propose an Autoregressive Maskable Proximal Policy Optimization (PPO) approach. Policy-based methods such as PPO [9] efficiently handle large joint action spaces by decomposing them into robot-specific action spaces and computing factorized policies sequentially with action masking [10]. This decomposition mitigates the combinatorial explosion of actions while enabling stable policy updates. The autoregressive structure allows each robot's policy to condition on previous robots' actions, capturing key dependencies in multi-robot coordination. Unlike value-based methods such as DQN [11], which require separate Q-functions per robot-action pair, our method uses a single policy network with integrated action masking for efficient execution.

Our approach generalizes well across diverse tool architectures and robot configurations, enhancing scalability and adaptability in semiconductor manufacturing. Ultimately, it aims to maximize throughput, measured by Unit Per Equipment Hour (UPEH) [12].

II. Related Work

Cluster tools in semiconductor manufacturing are primarily classified into radial-type and track-type structures, as described by Lee et al. [4]. Scheduling optimization for radial-type tools has been extensively studied. Rostami and Hamidzadeh [8] introduced an optimal periodic scheduler for dual-arm robots with residency constraints. Kim et al. [7] analyzed dual-armed cluster tool scheduling under strict time constraints, clarifying feasible processing time ranges. Chan et al. [6] investigated optimal scheduling for two-cluster configurations with constant robot moving times.

For multi-cluster tools, Zhu et al. [13], [14] proposed a virtual-wafer strategy for condition-based cleaning, reducing throughput losses during prolonged maintenance cycles [13]. They also developed a two-backward sequence strategy for single-arm tools with one-space buffer modules, achieving efficient wafer coordination and lower-bound cycle times [14].

DRL has also been explored for cluster tool scheduling. Hong and Lee [15] proposed a multi-agent RL approach for condition-based chamber cleaning. Kim and Lee [16] combined DRL with adaptive search strategies for concurrent wafer processing without predefined cyclic schedules. Lee and Lee [17] demonstrated that simple Deep Q-Network (DQN) methods can effectively manage radial-type cluster tools without extensive preprocessing.

While many previous studies have addressed single-arm and dual-arm configurations, this study focuses on multi-robot scheduling with flexible robot coordination. Our approach is tested in a radial-type cluster tool with 2 robots and a track-type cluster tool with 9 robots, covering a range of scales and providing a basis for future studies on larger systems.

Additionally, this study deliberately excludes equipment-dependent factors such as wafer residency time, chamber cleaning cycles, equipment failures, and maintenance, which introduce additional complexities that are beyond the current scope.

III. PRELIMINARIES

A. Semiconductor Cluster Tools

Semiconductor cluster tools are automated equipment systems commonly employed in front-end semiconductor processes such as photolithography, etching, and cleaning. These tools consist of multiple processing chambers arranged around one or more transport robots in a cluster-based architecture. Efficient and accurate wafer transport is critical for maintaining processing quality and achieving high throughput.

To evaluate productivity, a widely used industry metric is *Units Per Equipment Hour (UPEH)*, defined as:

$$UPEH = \frac{\text{Processed Units}}{\text{Active Equipment Hours}} \tag{1}$$

This metric, originally introduced in semiconductor equipment patents [12], has become a standard for measuring throughput and equipment utilization. In this study, reinforcement learning is applied with the explicit goal of maximizing UPEH through optimal robot scheduling policies.

Each wafer is assigned a predefined sequence of processing steps, known as a *flow recipe*. Every stage in the recipe is referred to as a *waypoint*, and wafers must pass through all waypoints in order before returning to the LoadPort to complete their processing cycle.

Only robots are responsible for wafer transport. PMs and airlocks cannot move wafers autonomously. Robots may have one or more arms, each carrying only one wafer at a time. Importantly, robot actions are sequential: if one arm is active, the other arms on the same robot must remain idle.

All PMs are limited to handling one wafer at a time. Additionally, the simulation environment assumes a single LoadPort with an effectively unlimited wafer supply, and does not model cassette replacement to simplify the setup.

Robot movement speed and the physical layout of modules determine travel time between locations. These structural constraints—combined with diverse chamber configurations and flow recipes—result in a challenging scheduling problem that requires careful coordination for throughput optimization.

B. Reinforcement Learning

We formulate the robot scheduling problem as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, r, P, \gamma \rangle$ with finite state space $\mathcal{S}$, finite action space $\mathcal{A}$, transition function P, reward function r, and discount factor γ. At each discrete timestep t, an agent observes $s_t \in \mathcal{S}$, takes action $a_t \in \mathcal{A}$, and receives a reward $r_t = r(s_t, a_t)$. The objective is to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{A})$ that maximizes the expected cumulative rewards $J^\pi = \mathbb{E}[\sum_{t=0} \gamma^t r_t | P, \pi]$, where Δ denotes a probability simplex over a set.

We assume the action space is factorizable with respect to the set of robots $\mathcal{R}$ with $|\mathcal{R}| = n$, such that $\mathcal{A} = \mathcal{A}_{r_1} \times \mathcal{A}_{r_2} \times \cdots \times \mathcal{A}_{r_n}$, where $\mathcal{A}_{r_i}$ is the i-th robot's action space. Here, we decompose joint action selection into a sequence of individual robot decisions in an autoregressive manner:

$$\begin{aligned} \pi(a|s) &= \pi(a_{r_1}, a_{r_2}, \ldots, a_{r_n}|s) \\ &= \pi(a_{r_1}|s) \cdot \pi(a_{r_2}|s, a_{r_1}) \\ &\quad \cdot \ldots \cdot \pi(a_{r_n}|s, a_{r_1}, \ldots, a_{r_{n-1}}), \end{aligned} \tag{2}$$

where $a_{r_i} \in \mathcal{A}_{r_i}$ is the i-th robot's action and we denote robot-specific policies as π for simplicity. This can be viewed as a form of semi-MDP [18], where the joint action a is an option and a_{r_i} are the corresponding primitive actions.

For our learning algorithm, we adopt Proximal Policy Optimization (PPO) [9], a policy gradient method known for its stable performance across various domains. PPO uses a clipped surrogate objective function:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where $r_t(\theta)$ is the probability ratio of the new and old policies, ϵ is the clip range hyperparameter, and $\hat{A}_t$ is an estimated advantage.

Policy gradient algorithms are particularly effective for our heterogeneous multi-robot setup as they naturally support action masking [10]. This mechanism eliminates invalid actions by zeroing out their logits before the softmax operation, allowing efficient implementation of robot-specific policies with a single policy network.

IV. PROBLEM STATEMENT

Transport robots in semiconductor equipment operate with discrete commands, such as moving to predefined PM positions and performing pick or place actions. However, as the number of robots and PMs increases, the action space expands rapidly, complicating scalable scheduling. This challenge is further amplified by the structural diversity of cluster tools. For example, track-type systems, commonly used in photolithography and cleaning, feature vertically stacked PMs and rail-guided robots, resulting in complex transport paths and repeated wafer flows.

Traditional multi-robot algorithms, commonly used in warehouse automation, delivery systems, and exploration, are difficult to apply directly in semiconductor manufacturing. A major challenge is estimating task duration, which varies significantly with equipment conditions and recipes. For instance,

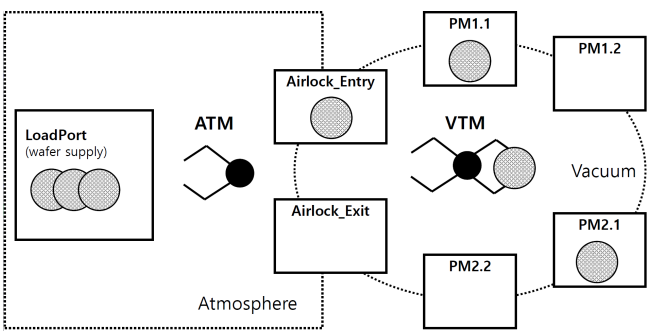

Fig. 1: Environment A Layout (Radial-Type)

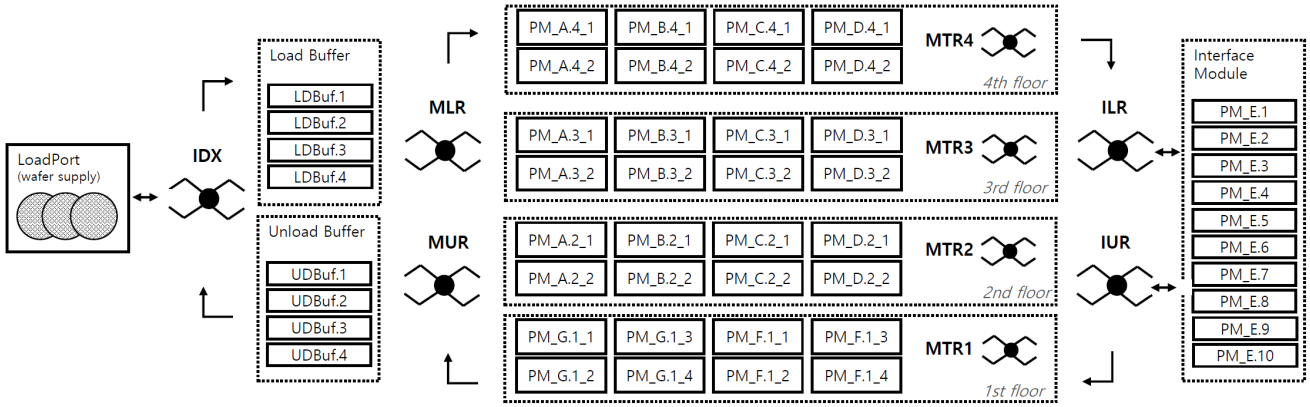

Fig. 2: Environment B Layout (Track-Type). See Table I for detailed configurations.

the time required for heating or cooling a wafer depends on the PM's current temperature, which fluctuates during operation. Additionally, robots in semiconductor tools typically operate in fixed regions with minimal overlap, unlike more flexible task-sharing systems. These constraints limit the applicability of conventional cooperative task allocation strategies.

Due to these domain-specific challenges, the semiconductor industry often relies on heuristics based on expert knowledge and process data. While effective in fixed setups, these methods lack adaptability to changing tool configurations and dynamic conditions. To address this, we evaluate a learning-based approach using two simulation environments representing common cluster tool architectures.

A. Dual-Robot Vacuum Cluster (Environment A)

The first equipment configuration is based on a radial-type cluster tool previously studied using a DQN-based scheduling approach [17]. We retain the overall layout and module structure from that study, but adjust robot speeds and processing times to better reflect realistic operating conditions. The system includes two robots operating in separate pressure domains: a Vacuum Transport Module (VTM) and an Atmospheric Transport Module (ATM). The robot layout and module connectivity are shown in Figure 1.

Modules labeled `PM1.x` and `PM2.x` are interchangeable within each recipe step, performing the same type of processing. The `LoadPort` provides infinite wafer supply for simulation simplicity. Two airlocks transfer wafers between domains: `Airlock_Entry` starts at atmospheric pressure and becomes vacuum after receiving a wafer from the ATM, while `Airlock_Exit` performs the reverse. The ATM has a single arm, while the VTM has two arms that can each hold a wafer, but must pick and place sequentially. Both robots move at 1000 mm/s, and travel time is based on module distance.

B. Track-Type Multi-Robot Cluster (Environment B)

The second environment models a cluster tool with industrial-level complexity, inspired by track-type systems commonly used in photolithography and cleaning. Unlike radial vacuum cluster tools, this system operates under atmospheric conditions and features vertically stacked modules across four physical floors. It also incorporates a simplified interface subsystem that mimics wafer exchange with external equipment. The overall system layout is shown in Figure 2.

Wafer transport is managed by nine dual-arm robots, each assigned to a specific region or transfer task. The main transfer robots (`MTR1` to `MTR4`) handle wafer transport within each floor. The `MLR` (Main Loading Robot) loads wafers into the system, while the `MUR` (Main Unloading Robot) removes them. The `ILR` (Interface Loading Robot) and `IUR` (Interface Unloading Robot) handle inter-floor transfers via the interface unit.

Each module is named based on its function, floor level, and index on that floor. For example, `PM_A.3_2` refers to an A-type PM located on the 3rd floor and represents the second instance of that type on that level. The letter `A` denotes an abstracted processing function, such as coating, heating, or cooling, which may vary by system configuration.

Details of robot roles and module configurations for both Environment A and Environment B are illustrated in Table I.

V. Proposed Methodology

A. Simulation and Learning Framework

Our simulation framework is fully configurable through a JSON-based input file, enabling users to define processing modules, robot configurations, individual robot action sets, flow recipes, and explicit deadlock conditions. A deadlock is defined as a system state in which wafers cannot proceed along their designated flow recipes due to resource conflicts or blocked paths. These states are pre-specified in the configuration file, and during execution, the simulator applies action masking to prevent transitions that would lead to such deadlocks. This design enables testing in diverse environments without modifying the simulation code.

B. Observation and Action Space

a) Observation Space: The observation vector is constructed by concatenating feature vectors from all process modules and transport robots, along with the current simulation time. Each feature is normalized to the range [0, 1], except for boolean fields, which are retained as binary indicators.

Environment A					Environment B				
Module	**Process Time (s)**	**Waypoint**	**Robot**	**Transport Path**	**Module**	**Process Time (s)**	**Waypoint**	**Robot**	**Transport Path**
LoadPort	0.0	[1, 6]	ATM	LoadPort → Airlock_Entry	LoadPort	0.0	[1, 11]	IDX	LoadPort → LDBuf.*
Airlock_Entry	10.0	[2]		Airlock_Exit → LoadPort	LDBuf.*	0.0	[2]		UDBuf.* → LoadPort
Airlock_Exit	10.0	[5]	VTM	Airlock_Entry → PM1.*	UDBuf.*	0.0	[10]	MLR	LDBuf.* → PM_A.*
PM1.*	45.0	[3]		PM1.* → PM2.*	PM_A.*	60.0	[3]	MUR	PM_G.* → UDBuf.*
PM2.*	65.0	[4]		PM2.* → Airlock_Exit	PM_B.*	60.0	[4]	MTR1-4	Transport assigned to each floor
					PM_C.*	60.0	[5]	ILR	PM_D.* → PM_E.*
					PM_D.*	60.0	[6]	IUR	PM_E.* → PM_F.*
					PM_E.*	100.0	[7]		
					PM_F.*	30.0	[8]		
					PM_G.*	30.0	[9]		

TABLE I: Module and robot transport configuration for Environment A and B, with process times including ±10% random variation to reduce overfitting and promote generalization.

Module features include remaining processing time, wafer presence, availability for pick/place, and current waypoint index. These features are listed in Table II.

TABLE II: Module Observation Features

Component	Size
Remaining processing time (float)	1
Current wafer's waypoint index in the flow recipe (float)	1
Flag indicating wafer is ready for pick (binary)	1
Flag indicating module can accept wafer (binary)	1
Wafer index in the module (float)	1
3D position of the module (float)	3

TABLE III: Robot Observation Features

Component	Size
Flag indicating whether action is already selected (binary)	1
Time remaining for current task (float)	1
Target action type (one-hot)	4
Target module position (float)	3
Current robot position (float)	3
Arm i ready to pick wafer (binary)	2
Arm i holds wafer ready to place (binary)	2
Waypoint index of wafer held by arm i (float)	2
Wafer index held by arm i (float)	2

Robot features include an initial flag indicating whether an action has been selected in the current timestep. This is followed by the selected target action, represented as a one-hot encoded vector specifying the command type (e.g., `move`, `pick`) and the spatial coordinates of the target module. Additional features include the current position, arm status, and task progress. Table III summarizes all robot-related features.

The final observation vector includes:

- Module features: $n_{\text{module}} \times d_{\text{module}}$
- Robot features: $n_{\text{robot}} \times d_{\text{robot}}$
- Simulation time: scalar value

These components are concatenated into a single 1D vector, where n_{module} and n_{robot} are the number of modules and robots, and d_{module} and d_{robot} are their respective feature dimensions. The resulting observation sizes for each environment are summarized in Table IV.

b) Action Space: The action space is a predefined discrete set, automatically generated by the framework based on the user-defined configuration of each simulation environment. It contains all valid action tuples that each robot may execute.

TABLE IV: Observation Vector Size per Environment

	Env A	**Env B**
Module Observation Size	56 (7 modules)	408 (51 modules)
Robot Observation Size	40 (2 robots)	180 (9 robots)
Time Scalar	1	1
Total Observation Size	**97**	**589**

Each action is represented as a tuple: `(robot, command, arm_index, target)`. The `command` is one of four types: `no_op`, `pick`, `place`, or `move`. When `arm_index` is 0, the action applies to the entire robot (e.g., `move`, `no_op`). For `pick` and `place`, the index specifies which arm is used. The `target` denotes the module the robot interacts with.

A sample of the generated action list is shown in Table V.

TABLE V: Sample Actions in Environment A

ID	Robot	Command	Arm	Target
0	ATM	no_op	0	
1	ATM	pick	1	LoadPort
2	ATM	pick	1	Airlock_Exit
3	ATM	place	1	LoadPort
⋮	⋮	⋮	⋮	⋮
8	VTM	no_op	0	
9	VTM	pick	1	Airlock_Entry
⋮	⋮	⋮	⋮	⋮
34	VTM	move	0	Chamber2.2

C. Multi-Robot Autoregressive Action Selection

This section details the implementation of our autoregressive (AR) action selection mechanism for multi-robot scheduling. At its core, at each timestep, the controller sequentially selects individual robot's actions in $n = |\mathcal{R}|$ substeps. Formally, the factorized policy (Eq. 2) requires robot-specific policies $\pi(a_{r_i}|s, a_{r_1}, \cdots, a_{r_{i-1}})$ for $1 \leq i \leq n$. Let us denote the aggregated condition at a timestep t and a substep k as $s_t^k := f^k(s_t, a_{t,r_1}, \ldots, a_{t,r_{k-1}}) \in \mathcal{S}'$, where $f^1 : \mathcal{S} \mapsto \mathcal{S}'$ and $f^k : \mathcal{S} \times \mathcal{A}^1 \times \cdots \times \mathcal{A}^{k-1} \mapsto \mathcal{S}'$ for $1 < k \leq n$ are aggregation functions which combines the current observation

TABLE VI: Per-robot action counts in Environment A and B.

Env A		Env B	
Robot	**#Actions**	**Robot Type**	**#Actions**
IDX	8	IDX	30
VTM	27	MLR + MUR	28 (each of 2 robots)
		MTR1–4	31 (each of 4 robots)
		ILR + IUR	46 (each of 2 robots)

s_t and previous actions $a_{t,r_1}, \ldots, a_{t,r_{k-1}}$ into a single vector s_t^k. Then at t, the control procedure becomes:

$$\begin{aligned} a_{t,r_1} &\sim \pi(\cdot|f^1(s_t)) \\ a_{t,r_2} &\sim \pi(\cdot|f^2(s_t, a_{t,r_1})) \\ &\vdots \\ a_{t,r_n} &\sim \pi(\cdot|f^n(s_t, a_{t,r_1}, \cdots, a_{t,r_{n-1}})) \\ a_t &= (a_{t,r_1}, \cdots, a_{t,r_n}). \end{aligned}$$

In our implementation, $f^1(s_t) = s_t$ and the aggregation function f^k for $k > 1$ updates corresponding fields in the last aggregated condition vector s_t^{k-1} using the previously selected action $a_{t,k-1}$, as detailed in Section V-B. This incremental update approach maintains computational efficiency while preserving all necessary information for subsequent decision steps.

We propose an efficient and straightforward way to implement robot-specific policies $\pi(a_{r_i}|s_t^k)$ using action masking approach [10]. We first define the union of all individual action spaces $\mathcal{A}_{\text{total}}$:

$$\mathcal{A}_{\text{total}} = \bigcup_{i=1}^{n} \mathcal{A}_{r_i}.$$

Then with a single parameterized function $\phi_\theta : S' \mapsto \mathbb{R}^{|\mathcal{A}_{\text{total}}|}$, we define our parameterized robot-specific policy π_θ as

$$\pi_\theta(\cdot|s_t^k) = \text{softmax}(\text{mask}^k(\phi_\theta(s_t^k))),$$
$$\text{mask}^k(l)_i = \begin{cases} l_i & \text{if } a_i \in \mathcal{A}_{r_k} \\ -\infty & \text{otherwise.} \end{cases}$$

Therefore, our approach efficiently handles the combinatorially large joint action space with a single policy network in an autoregressive manner. Note that the policy network only need to output $\mathcal{A}_{\text{total}}$ logits at each step, enabling a compact model. Furthermore, this method allows us to easily impose additional constraints by modifying the mask function to exclude actions violating the constraints.

A quantitative comparison of per-robot action counts in *Environment A* and *Environment B* is provided in Table VI, while the resulting decision complexity under the joint-action model and our autoregressive formulation is summarized in Table VII. These tables highlight the exponential growth of the joint action space—particularly in *Env B*—and demonstrate how our approach significantly reduces the number of candidate actions by decomposing decisions sequentially across robots.

TABLE VII: Decision complexity and action space size comparison between joint and AR (ours) models.

	Joint	**AR (Ours)**
Decision Complexity	$\mathcal{O}\left(\prod_i \lvert\mathcal{A}_{r_i}\rvert\right)$	$\mathcal{O}\left(\sum_i \lvert\mathcal{A}_{r_i}\rvert\right)$
Env A Action Count	216	35
Env B Action Count	4.4×10^{13}	302

D. Action Masking for Feasibility Filtering

To reduce the effective action space during sequential decision-making, we apply action masking to restrict the policy to only feasible actions at each step. Unlike prior work [17] that penalizes invalid actions post-selection, our method eliminates them upfront through masking.

At each robot's turn, a mask is applied to its action space $\mathcal{A}_{r_i} \subset \mathcal{A}_{\text{total}}$, enabling only valid actions based on the current environment state. This avoids wasting exploration on infeasible options and significantly reduces policy complexity. The overall procedure is outlined in Algorithm 1, where the agent filters infeasible actions using robot status and environment constraints before selecting each action.

Algorithm 1 Autoregressive Action Selection with Dynamic Masking

while there exist robots without assigned actions **do**
  $r_{\text{curr}} \leftarrow$ next robot to select an action
  $\mathcal{A}_{\text{total}} \leftarrow \bigcup_i \mathcal{A}_{r_i}$ ▷ Union of all robot action sets
  $M \leftarrow$ initialize mask of size $|A_{\text{total}}|$ with all zeros
  for each action $a \in \mathcal{A}_{\text{total}}$ **do**
    if $a \notin \mathcal{A}_{r_{\text{curr}}}$ **then**
      $M[a] \leftarrow 1$ ▷ Mask actions of other robots
    else if r_{curr} is busy **and** $a \neq$ `no-op` **then**
      $M[a] \leftarrow 1$ ▷ If busy, allow only `no-op`
    else if a is invalid in current state **then**
      $M[a] \leftarrow 1$ ▷ Mask infeasible actions
    end if
  end for
  Select $a_{\text{curr}} \sim \pi(\cdot|s, M)$
  Update observation s based on a_{curr}
end while
Execute all selected actions simultaneously
Advance simulation time by 1 second

Implementation note: To reduce runtime overhead, we cache action masks using hash keys derived from discrete observation features (e.g., wafer presence, robot status). The hash table is size-limited and managed as a circular buffer to support fast reuse.

E. Reward Design

Designing an effective reward function is essential for guiding learning in complex scheduling environments. Sparse and delayed reward signals are common in reinforcement learning. In our setting, this issue becomes more severe due to long time horizons and delayed wafer completions, which make credit assignment across timesteps particularly difficult. To address this, we define a reward function aligned with the production goal of maximizing *Units Per Equipment Hour (UPEH)* by

combining three reward components: wafer completion, step-wise progress, and movement efficiency.

a) Completion Reward: A reward of $+1$ is given when a wafer completes its entire process and returns to the load port. While this directly aligns with maximizing throughput, it is sparse and insufficient for efficient learning when used alone.

b) Wafer Progress Reward: To provide more frequent learning signals, we introduce a per-step reward based on wafer progress.

$$\begin{aligned} w_m &= \sum_{i=1}^{k_m} (p_{i-1} + d_i) \\ P_t &= \sum_{m \in \mathcal{M}} (w_m + \tau_m) \\ r_t^{\text{progress}} &= P_t - P_{t-1} \end{aligned} \tag{3}$$

Each module m is assigned a **progress constant** w_m, which is computed once from the wafer flow recipe at the beginning of training. Here, k_m denotes the *waypoint index* of module m in the process flow sequence. The summation index i iterates over all preceding waypoints, where p_i is the *processing time* at step i, and d_i is the *transport time* to the next step. Thus, w_m represents the **ideal accumulated time** required for a wafer to reach module m under perfect conditions with no waiting or blocking delays.

At each timestep t, the total wafer progress P_t is computed by summing the target constants w_m and the elapsed processing times τ_m for wafers currently in each module m, where τ_m denotes the *elapsed time* that a wafer has been processed in module m so far. Since w_m reflects the ideal time to reach module m, it remains unchanged regardless of actual delays. As a result, the agent is encouraged to minimize delays in order to accumulate greater progress rewards within fewer steps.

c) Idle Move Penalty: Robots are penalized for inefficient motion patterns in which movement is not followed by meaningful interaction. Specifically, a small penalty is applied when move actions are executed consecutively without an intervening pick or place, or when the subsequent pick/place occurs in a module different from the target of the preceding move. This design discourages unnecessary travel and promotes goal-directed motion.

TABLE VIII: Training and evaluation settings.

Parameter	Value	Parameter	Value
Batch size	256	Discount factor γ	0.995
Total timesteps	3M	Learning rate	5e-5
Num. of envs	20	PPO clip range	0.2
Rollout steps	256	GAE lambda	0.95
PPO epochs	10	Network (shared)	2-layer MLP (256 units)

Reward weights:
$k_1 = 1.0$, $k_2 = 0.01$, $k_3(EnvA, B) = 0.1, 0.2$
Time step: Training = 1.0s, Evaluation = 0.1s
Episode length: Training = 1000s, Evaluation = 3600s

Final Composition. The final reward is computed as a weighted sum of the three components:

$$r_t = k_1 \cdot r_t^{\text{completion}} + k_2 \cdot r_t^{\text{progress}} - k_3 \cdot r_t^{\text{idle}} \tag{4}$$

where k_1, k_2, and k_3 are tunable weights for balancing the reward terms (see Table VIII for values).

VI. Experiments

A. Experimental Setup

We design two configurable environments representing typical cluster tool structures: Environment A and B. To prevent agents from overfitting to fixed processing patterns, each PM is assigned a stochastic processing time with $\pm 10\%$ random variation. Environments are initialized in a fully-loaded state to ensure consistent wafer availability and accurate throughput measurement.

We train agents using the Maskable PPO algorithm from Stable-Baselines3 [10], [19]. Unlike prior work [17], which defined episodes by wafer count, we use time-based episodes with fixed maximum durations. This approach better reflects continuous, high-throughput operations, where wafers are continuously fed without clear episode boundaries. Episodes are time-limited during training and evaluation to align with UPEH measurement and ensure consistent reward assessment. Table VIII summarizes the full hyperparameter settings.

The simulator advances in one-second discrete time steps during training, balancing simulation fidelity and state diversity. Finer-grained steps (e.g., millisecond-level) produced many near-identical consecutive states, degrading learning performance [20]. For evaluation, we use millisecond-level deterministic rollouts for accurate throughput measurement.

TABLE IX: Comparison of UPEH (wafers per hour) between a rule-based heuristic schedule and our method. We indicate 95% confidence intervals across 5 independent runs. All UPEH values are rounded up to the nearest integer.

Environment	Ours (UPEH)	Heuristic (UPEH)
Env A	96 ± 5	58
Env B	304 ± 1	276

B. Training Performance in Two Environments

Figure 3 presents the training performance metrics of our proposed model in both environments. The average episodic reward increases steadily and converges within the first 30–50% of total training steps, indicating stable learning dynamics. This trend is consistent across both *Environment A* and *Environment B*, despite the increased complexity of the latter.

To further understand learned scheduling behavior, we visualize robot and PM activity using Gantt charts. As shown in Figure 4, PMs are underutilized and wafer processing is irregular early in training. After training, however, the schedule becomes densely packed with minimal idle time, reflecting the agent's improved coordination and process awareness.

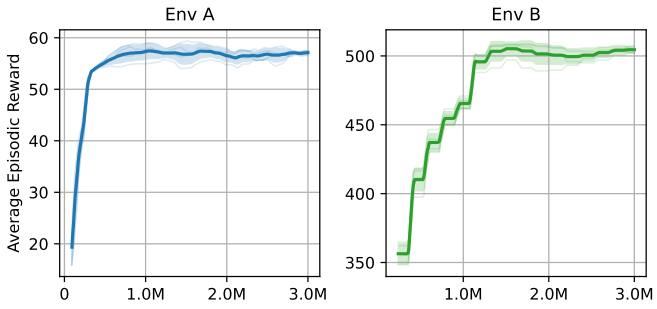

Fig. 3: Average episodic reward during training. **Left:** Env A (Radial-Type), **Right:** Env B (Track-Type). The x-axis indicates training steps. Each curve represents the average over 5 independent runs, with shaded areas indicating standard deviation.

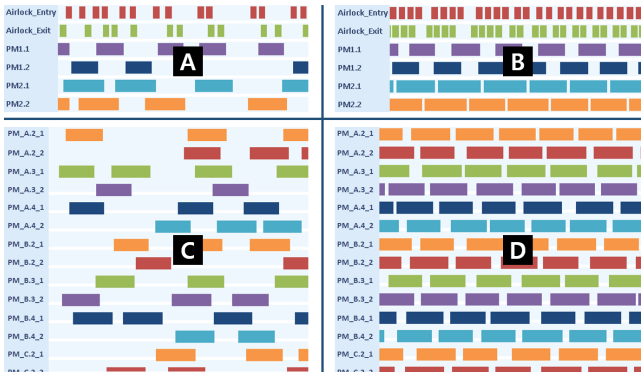

Fig. 4: Gantt chart visualizations of PM activity before and after training. (A) Env A before training, (B) Env A after training, (C) Env B before training, (D) Env B after training. Irregular scheduling is replaced by densely packed wafer processing after training. Charts are generated using a custom analysis tool built on simulator logs.

C. Rule-Based Scheduling Reference

We implement a rule-based heuristic scheduler inspired by decision-making rules commonly used in industrial cluster tools. Robots prioritize loading over unloading whenever wafers are available, and in case of a tie, choose the nearest available module. The scheduler relies solely on current state information without predicting future module availability. Given the distinct roles of robots and the need to avoid deadlocks, the logic is implemented separately for each environment. For fair comparison, the same feasibility constraints as our method were applied.

This heuristic serves as a straightforward baseline, capturing the core decision-making logic of traditional manufacturing systems. While it lacks the predictive depth and flexibility of learning-based models, it provides a meaningful comparison, highlighting the advantages of data-driven scheduling. As shown in Figure 5, the heuristic scheduler exhibits repetitive cyclic behavior in both environments. In contrast, our learning-based policy produces more adaptive and efficient schedules, as illustrated in Figure 4. This qualitative difference is also reflected in UPEH performance: as shown in Table IX, our method consistently outperforms the heuristic reference in both environments.

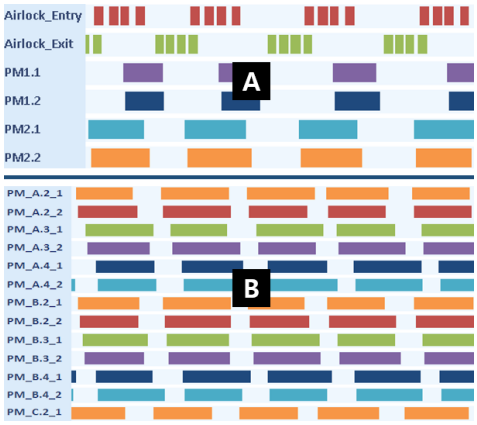

Fig. 5: Gantt charts of PM activity under the heuristic scheduler. (A) Env A and (B) Env B both exhibit cyclic scheduling patterns, in contrast to the more adaptive behavior learned by the RL agent.

TABLE X: UPEH under different reward settings. We indicate 95% confidence intervals across 5 independent runs. All UPEH values are rounded up to the nearest integer.

Environment	Completion Only	Progress Only	Both Combined
Env A	81 ± 3	91 ± 6	96 ± 5
Env B	52 ± 5	301 ± 2	304 ± 1

D. Ablation Study on Reward Design

We examine the impact of each reward component in the final reward formulation (Equation 4). The reward function is designed to balance short-term progress and long-term task completion, capturing the sequential, multi-step structure of wafer transfer, where each action contributes to a final outcome over a long horizon.

An ablation study was conducted to compare three configurations: (1) only the Completion Reward, (2) only the Progress Reward, and (3) both combined. As shown in Table X, using only the Completion Reward resulted in significantly lower UPEH, particularly in complex environments like Env B, where sparse, long-term rewards alone were less effective due to delayed feedback. In contrast, the Progress Reward provided more frequent signals, promoting steady task progression. Combining both rewards produced the highest UPEH, as the dense signals from the Progress Reward complemented the long-term focus of the Completion Reward, supporting more balanced task scheduling, which is critical for high system performance.

E. Penalty and Masking for Idle Movement

To reduce unnecessary robot movement, we apply a negative penalty to idle motion, defined by the $-k_3 \cdot r_t^{\text{idle}}$ term in Eq. 4. This penalty discourages idle movements that are not directly linked to pick or place actions, encouraging the robot to preemptively move to the expected processing module in advance. By promoting coordinated movement, this approach minimizes isolated move commands that do not contribute to immediate task execution, reducing overall travel distance.

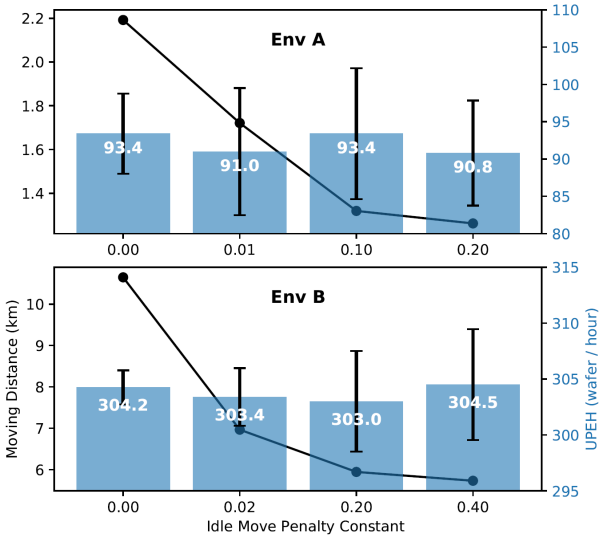

Fig. 6: Effect of k_3 (idle move penalty) on robot behavior. Average movement distance is shown as a solid line, while mean UPEH is represented as bars, including 95% confidence intervals based on successful runs.

As k_3 increased, movement distance dropped sharply, indicating effective penalty enforcement (Figure 6). However, throughput remained largely stable, even with strong penalties. Notably, larger k_3 values led to significant training instability, causing some runs to fail completely, with wide performance variance and occasional catastrophic collapse.

VII. Conclusion and Future Work

We present a reinforcement learning framework that effectively learns wafer transfer scheduling policies for semiconductor cluster tools. The proposed method, based on autoregressive action modeling and action masking, achieved scalable performance across two representative tool configurations, demonstrating practical applicability at the cluster tool level.

Despite these promising results, some limitations remain. The learned policies may still exhibit unnecessary idle movements, which, although tolerable in simulation, can pose challenges in industrial environments where predictable behavior is preferred. Additionally, the autoregressive formulation requires sequential environment steps during training, introducing simulator dependence and uneven discounting across decisions, as the reward for earlier actions is discounted more heavily than that for later actions within the same sequence.

Future work includes improving motion efficiency and exploring architectures that can capture temporal dependencies, such as RNNs, attention mechanisms, or memory-augmented models. These methods can leverage memory mechanisms to capture prior actions without directly including them in the observation state, improving long-term decision-making.

We also plan to refine the heuristic baseline, incorporating more sophisticated rule sets or optimization algorithms to provide a more meaningful comparison against learning-based approaches.

We release our implementation, including training code, environment definitions, and a Gantt chart visualization UI, at https://github.com/splendidz/ar_drl_cluster_tool to support reproducibility and further research.

References

[1] C. Pan, M. Zhou, Y. Qiao, and N. Wu, "Scheduling cluster tools in semiconductor manufacturing: Recent advances and challenges," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 586–601, 2018.

[2] J. Hwang and S. d. Noh, "Digital twin-based optimization of operational parameters for cluster tools in semiconductor manufacturing," *IEEE Access*, vol. 12, pp. 122 078–122 100, 2024.

[3] T. Perkinson, P. McLarty, R. Gyurcsik, and R. Cavin, "Single-wafer cluster tool performance: an analysis of throughput," *IEEE Transactions on Semiconductor Manufacturing*, vol. 7, no. 3, pp. 369–373, 1994.

[4] T.-E. Lee, "A review of scheduling theory and methods for semiconductor manufacturing cluster tools," in *2008 Winter Simulation Conference*, 2008, pp. 2127–2135.

[5] T.-E. Lee, H.-J. Kim, and T.-S. Yu, *Semiconductor Manufacturing Automation*. Cham: Springer International Publishing, 2023, pp. 841–863.

[6] W. K. V. Chan, J. Yi, and S. Ding, "Optimal scheduling of multicluster tools with constant robot moving times, part i: Two-cluster analysis," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 1, pp. 5–16, 2011.

[7] J.-H. Kim, T.-E. Lee, H.-Y. Lee, and D.-B. Park, "Scheduling analysis of time-constrained dual-armed cluster tools," *IEEE Transactions on Semiconductor Manufacturing*, vol. 16, no. 3, pp. 521–534, 2003.

[8] S. Rostami, B. Hamidzadeh, and D. Camporese, "An optimal periodic scheduler for dual-arm robots in cluster tools with residency constraints," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 609–618, 2001.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[10] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *The International FLAIRS Conference Proceedings*, vol. 35, 2022.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, and et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[12] J. Lee, "Semiconductor manufacturing equipment and method for calculating unit time yield," Korean Patent Patent KR100 859 436B1, Sep 9, 2008.

[13] Q. Zhu, H. Li, C. Wang, and Y. Hou, "Scheduling a single-arm multicluster tool with a condition-based cleaning operation," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 10, pp. 1965–1983, 2023.

[14] Q. Zhu, G. Wang, N. Wu, Y. Qiao, Y. Hou, M. Zhou, and S. Zhao, "Scheduling single-arm multicluster tools for two-type wafers with lower-bound cycle time," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 11, pp. 6658–6671, 2023.

[15] C. Hong and T.-E. Lee, "Multi-agent reinforcement learning approach for scheduling cluster tools with condition based chamber cleaning operations," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 885–890.

[16] H.-J. Kim and J.-H. Lee, "Scheduling cluster tools for concurrent processing: Deep reinforcement learning with adaptive search," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 3783–3796, 2025.

[17] C. Lee and S. Lee, "A practical deep reinforcement learning approach to semiconductor equipment scheduling," in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, vol. 1, 2021, pp. 979–985.

[18] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[19] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[20] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1904.12901