Scheduling Dual-Arm Cluster Tools With Multifunctional Process Modules Using Deep Reinforcement Learning

QingHua Zhu, *Senior Member, IEEE*, LangJin Liu, WeiXin Liang, and Yan Hou

***Abstract*—With the advancement of semiconductor manufacturing technology, multifunctional process modules (MPMs) have been widely adopted in cluster tools to enhance production flexibility and efficiency by handling multiple operations concurrently. However, the MPMs lead to challenges in scheduling the robot due to a variety of configurations, complex robot action sequences, and regulating wafer postprocessing residency time. For scheduling such a dual-arm cluster tool (DACT) with MPMs, this article proposes a specific reinforcement learning-based scheduling method. We first develop an adaptive algorithm to generate all feasible MPM configurations. We then employ the masking technique and prioritize a replay experience buffer to improve the dueling double deep *Q*-network (D3QN), enabling it to train and identify scheduling strategies that minimize makespan and reduce wafer residency time under each configuration. We conduct experiments to demonstrate that the proposed method ensures high productivity, providing a robust and flexible scheduling solution for cluster tools in semiconductor manufacturing, and significantly enhances overall production performance.**

***Index Terms*—Cluster tools, deep reinforcement learning, scheduling, semiconductor manufacturing, wafer fabrication.**

NOMENCLATURE

α_i	Time required to fabricate a wafer at step i.
c	Time needed for the robot' simple swap.
j& j+1	Indices of two successive multifunctional steps.
k	Number of MPMs.
l	Number of SPMs.
m_i	Number of process modules at step i.
N_q^+	= {1, 2,..., q}, q is a positive integer.
N_q	= $N_q^+ \cup \{0\}$.
n	Number of steps for processing a wafer.
PM	Process modules in a cluster tool.
PS_i	PMs for step i.
R	Robot in a cluster tool.
S_i	ith processing step for processing a wafer.
Swap i	Wafer exchange operation between the PS_i.
μ	R's moving time between two successive steps.
λ	Time that R spends in unloading or loading a wafer.
δ_i	Longest time that a wafer can remain at PS_i.
τ_i	Sojourn time of a wafer at PS_i.
Π_{iL}	Shortest time for finishing a wafer at step i.
Π_{iU}	Longest time for finishing one wafer at step i.
Θ	Cycle time of a cluster tool.
ψ_1	Total activity time of R during a cycle.
Π	Workload of a cluster tool.
WFP	Wafer flow pattern denoted by (m_1, $m_2, \ldots, m_n$).

Received 26 October 2025; accepted 25 January 2026. This article was recommended by Associate Editor M. P. Fanti. *(Corresponding author: QingHua Zhu.)*

QingHua Zhu, LangJin Liu, and Yan Hou are with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China (e-mail: zhuqh@gdut.edu.cn; 2112305029@mail2.gdut.edu.cn; houyan@gdut.edu.cn).

WeiXin Liang is with Guangzhou Institute of Technology, Xidian University, Guangzhou 510555, China (e-mail: 25181214337@stu.xidian.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TSMC.2026.3659946.

Digital Object Identifier 10.1109/TSMC.2026.3659946

I. INTRODUCTION

WITH the increasing demand for semiconductor chips, the complexity and precision requirements of manufacturing processes have also been continuously rising. As efficient and precise manufacturing equipment, cluster tools significantly enhance manufacturing efficiency by integrating multiple single-wafer process modules (PMs) and efficient transport systems. A cluster tool comprises multiple single-wafer PMs, a wafer transport robot (either single-arm or dual-arm), and loadlocks (LLs) for loading and unloading wafer cassettes. Once a wafer cassette is loaded into LLs, the robot sequentially unloads the wafers and loads them into the PMs according to the specified processing order. After all operations are completed, the wafers are returned to LLs. Cluster tools can be categorized into single-arm cluster tools (SACTs) and dual-arm cluster tools (DACTs). As shown in Fig. 1, in an SACT, a single-arm robot transports wafers, whereas in a DACT, a dual-arm robot transports wafers. Due to the higher efficiency of dual-arm robots compared to

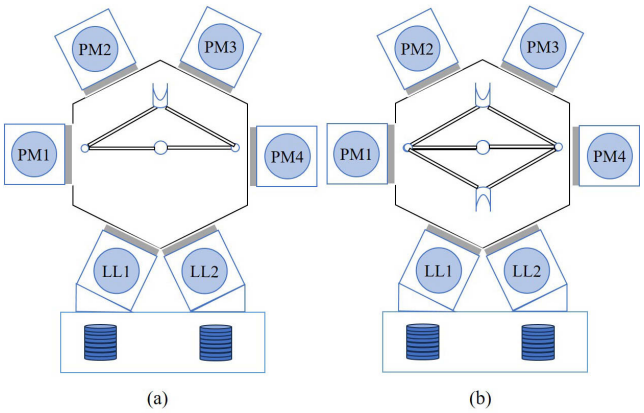

Fig. 1. Cluster tools. (a) Single-arm robot. (b) Dual-arm robot.

single-arm robots, DACTs are more frequently employed [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12].

The diverse nature of wafer fabrication processes results in varying requirements for cluster tools, depending on the specific processes involved. Wafer residency time constraints (WRTCs) require that a wafer be unloaded from the processing module within a particular time after processing. Failing to do so may result in wafer damage due to high temperatures or chemical residue. Such constraints are particularly prevalent in processes such as low-pressure chemical vapor deposition (LPCVD) and atomic layer deposition (ALD) [7], [13], [14], [15], [16]. For SACTs and DACTs with such constraints, [17] and [18], respectively, provide the necessary and sufficient conditions for schedulability, along with an optimal one-wafer periodic schedule in a steady state. Additionally, [19] employs timed Petri nets to analyze a DACT to find a noncyclic schedule.

Under WRTCs, cluster tools sometimes fabricate multiple types of wafers together. To address the challenge of efficiently processing multiple wafer types, [14] proposes a closed-form based algorithm for scheduling the processing of multiple wafer types in SACTs under WRTCs in a steady state. Additionally, [20] presents a method to determine the minimum cycle time for DACTs under the same constraints. Other approaches to address similar scheduling problems have also been studied [7], [11], [21], [22], [23], [24].

Processing a batch of wafers involves three stages: 1) start-up; 2) steady-state; and 3) close-down. The first and third stages are referred to as transient processes, during which unexpected errors are more likely to occur. The transient-state scheduling problem for DACTs is studied in [8], while [25] and [26] optimize the start-up and close-down processes of SACTs, respectively. In steady-state scheduling, cyclic schedules are commonly used due to their operational simplicity [14], [27], [28]. The cleaning operations for SACTs and DACTs with parallel chambers are studied in [29] and [9], respectively. Residual chemicals and impurities accumulate in the chambers during wafer processing, which, if not cleaned before reaching a certain threshold, can damage the wafers. Qiao et al. propose using virtual wafers to facilitate cleaning operations [30], while [31] addresses the same problem in multicluster tools. Although scheduling multicluster tools is more challenging, their higher production efficiency has motivated extensive research to develop effective scheduling solutions [8], [20], [21], [22], [31], [32], [33], [34], [35], [36]. Reinforcement learning has been introduced into scheduling cluster tools [37] and testing facilities [38] in semiconductor manufacturing, enabling adaptive, advanced functionality.

The above studies are all based on single-function PMs (SPMs). Since SPMs can perform only one operation at a time, recent research in the wafer fabrication industry has focused on multifunctional PMs (MPMs) to enhance equipment productivity and flexibility [32]. MPMs can perform multiple functions without requiring wafer transfer between operations, and they can also be configured to perform only one of the operations if needed. Coating cluster tools incorporating MPMs have been applied in semiconductor manufacturing for light-emitting diodes (LEDs). The coating process consists of four steps: wafer cleaning, photoresist coating, soft baking, and cooling. For the first two steps, MPMs can quickly switch between water supply and photoresist application, allowing consecutive operations with minimal switching time. The subsequent soft baking and cooling steps are carried out individually by corresponding SPMs. Therefore, the configuration of MPMs significantly impacts the efficiency of the coating cluster tools by influencing switching time and throughput.

However, scheduling DACTs with MPMs faces the following challenges.

1) Dynamic configuration complexity. The operational steps of MPMs must be dynamically adjusted according to the processing recipe. This flexibility significantly increases scheduling complexity, particularly in module switching and resource allocation. Dynamic optimization strategies are required to balance task-switching costs and system efficiency while satisfying real-time scheduling requirements.
2) A wide variety of robot sequences across three phases. A DACT with MPMs undergoes start-up, steady-state, and close-down phases, which require entirely different, coordinated robot actions among MPMs and SPMs to complete multiple processing steps, dynamic switching between modules, and parallel task execution. It is difficult to apply a single method to schedule the robot to finish three phases.
3) Biobjective optimization requirements. As the circuit width decreases, wafer processing technologies impose strict constraints on wafer residency time. Shortening the wafer postprocessing residency time is also crucial to ensure high-quality circuits. However, traditional methods often struggle to effectively optimize the makespan and wafer postprocessing residency time, which demands computational efficiency under complex system scales and dynamic process constraints.

To address the challenges above, this study achieves the following innovative contributions in both theory and practice.

1) We propose a scheduling algorithm, dueling double deep Q-network (D3QN) with a mask mechanism and prioritized experience replay (PER), named MD3QN.

By introducing the mask mechanism, the algorithm effectively reduces the exploration of invalid actions in large decision spaces, thus improving the convergence speed and stability of the training process. PER allows the algorithm to prioritize the reuse of high-priority experiences during training. PER is particularly effective for a cluster tool during transitions between steady and transient states, accelerating learning.

2) We propose a comprehensive scheduling optimization scheme encompassing the entire operational sequence from the start-up and steady-state stages to the close-down stage. This scheme can provide the complete sequence of robotic arm actions required to process a batch of wafers, aiming to minimize the makespan. The scheme effectively handles the different robot operation sequences among the three stages.
3) Dual objective optimization and trade-off scheme: throughout the scheduling process, in addition to minimizing the makespan, this study also emphasizes wafer postprocessing residency time. The scheme prioritizes reducing the wafer residency time in cases requiring trade-offs while accepting a nonminimal makespan. Through this dual-objective optimization strategy, the proposed method provides efficient and automated scheduling decisions for practical production scenarios, enhancing the system's flexibility and overall efficiency.

In Section II, we describe the problem. Section III presents an algorithm to find valid MPM patterns. Section IV details the D3QN reinforcement learning algorithm with masks and PER. Section V demonstrates how we apply the proposed methods to solve real-world examples. Finally, Section VI concludes this article.

II. Problem Description

Let $n > 2$ stand for the number of process steps in a DACT. An MPM can perform one or two functions (steps). Let j and $j + 1$ denote the indices of two successive multifunctional steps. Among these steps, consecutive steps j and $j + 1$ ($j < n$) are possible to be executed using MPMs, where j is specified by the engineer. Depending on the allocation strategy, the execution of steps j and $j + 1$ can follow two options.

1) *Combined Processing:* Steps j and $j + 1$ are executed together by the same MPM, forming a combined step denoted as $j\&j+1$. In this case, the two steps are treated as a single unit, and their execution sequence does not require separate scheduling.
2) *Independent Processing:* Steps j and step $j + 1$ are executed independently by two separate MPMs. The two steps are treated as distinct processes that require scheduling and resource allocation.

Except for steps j and $j + 1$, other steps are executed using SPMs whose processing is not affected by the allocation strategies of the multifunctional modules.

Let m_i denote the number of PMs at step i, where $i \in N_q^+ = \{1, 2, \ldots, q\}$ and q is a positive integer. Treating steps j and $j+1$ independently may enhance scheduling flexibility and enable more efficient resource utilization under certain operational conditions. Let $m_{j\&j+1}$ denote the number of MPMs assigned to process the combined step $j\&j + 1$. Note that m_i, $i \in \{j, j + 1\}$, indicates the number of MPMs allocated for step i.

For a DACT, let k stand for the number of MPMs and l stand for the number of SPMs, $k \in N_q^+\backslash\{1\}$, $l \in N_q^+\backslash\{1\}$). Allocating different MPMs to perform step j (m_j), step $j + 1$ (m_{j+1}), and step $j\&j+1(m_{j\&j+1})$ should adhere to the following two MPM-configuration conditions:

1) $m_j + m_{j+1} + m_{j\&j+1} = k$;
2) ($m_{j\&j+1} = k$ and $m_j = 0$ and $m_{j+1} = 0$) or ($0 \le m_{j\&j+1} < k$ and $m_j > 0$ and $m_{j+1} > 0$).

In this study, all MPMs must be fully utilized without any idleness; thus $m_j + m_{j+1} + m_{j\&j+1} = k$. Additionally, since m_j represents the number of MPMs allocated to step j, $m_j > 0$ indicates that at least one MPM is allocated to step j. In order to ensure process continuity, at least one MPM must be allocated to step $j+1$. Otherwise, after completing step j, no MPM would be available for step $j+1$, resulting in violating operational requirements. Similarly, if $m_{j+1} > 0$, m_j must also be positive; otherwise, the MPMs allocated to step $j+1$ would become idle.

Allocation patterns that fail to meet the rules above are considered invalid. Let $< m_{j\&j+1}, m_j, m_{j+1} >$ represent valid MPMs allocation patterns, where each element denotes the number of MPMs assigned to step $j\&j + 1$, step j, and step $j + 1$, respectively. We must identify all the valid patterns and select one for processing a batch of wafers, which is presented in Section III.

In semiconductor manufacturing, many processes require a wafer to be unloaded promptly after processing. For example, the photoresist coating step aims to apply the photoresist material uniformly onto the wafer surface. Due to the substantial solvent residue in the photoresist layer after coating, the diffusion of these solvents can adversely affect the dimensional accuracy and sidewall profile of the photoresist. Therefore, it is essential to promptly transport the wafer to the soft bake step to remove these solvents and cure the photoresist layer. Shortening the wafer postprocessing residency time is crucial to enhancing the stability of the photoresist layer.

Therefore, the scheduling objective is to optimize the makespan and wafer postprocessing residency time.

III. Allocating MPM to Process Steps

We propose Algorithm 1 to generate all valid patterns aforementioned. By inputting k, l, and j, the algorithm generates all valid patterns based on the specified constraints.

Let $k = 3$, $l = 4$, and $j = 1$ as examples. As shown in Fig. 2, after all invalid patterns are excluded, the remaining valid patterns are <3, 0, 0>, <1, 1, 1>, <0, 2, 1>, and <0, 1, 2>. Based on the different values of $m_{j,j+1}$, m_j, and m_{j+1}, the patterns can be classified into three cases:

1) $m_{j\&j+1} = k$ and $m_j = m_{j+1} = 0$;
2) $m_{j\&j+1} > 0$, $m_j > 0$, and $m_{j+1} > 0$;
3) $m_{j\&j+1} = 0$, $m_j > 0$, and $m_{j+1} > 0$.

For cases 1 and 3, the wafer flow patterns (WFPs) are $(m_1, \ldots, m_{j-1}, m_{j\&j+1}, m_{j+2}, \ldots, m_n)$ and $(m_1, \ldots, m_{j-1}, m_j,$

Algorithm 1 Generate Valid Patterns and WFPs

1. Input k, l, and j // j and $j+1$ are the indices of steps with MPMs
2. Initialize *patterns* and *WFPs*.
3. **For** $m_{j\&j+1} \leftarrow 0$ to k **do**.
4. **for** $m_j \leftarrow 0$ to $k - m_{j\&j+1}$ **do**.
5. $m_{j+1} \leftarrow k$–$m_{j\&j+1}$–m_j:pattern < $m_{j\&j+1}$, m_j, m_{j+1}>.
6. **If** MPM-configuration conditions hold Then
7. Add the pattern to *patterns*.
8. WFP ← (pattern, l).
9. Add the WFP to *WFPs*.
9. **END If**
10. **End For**
11. Output *patterns* and *WFPs*.

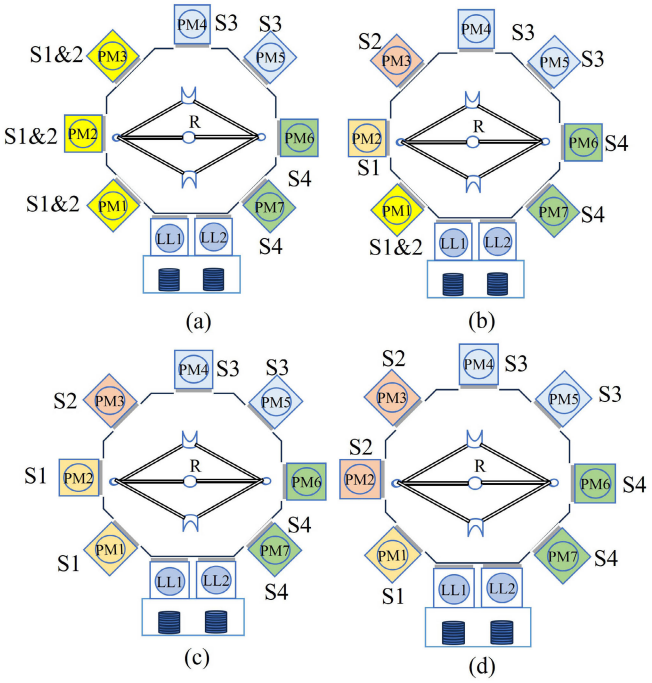

Fig. 2. Valid WFPs. (a) (3, 0, 0). (b) (1, 1, 1). (c) (0, 2, 1). (d) (0, 1, 2).

m_{j+1}, $m_{j+2}, \ldots, m_n$). In case 1, the sequence of wafer processing steps is $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_{j-1} \rightarrow S_{j\&j+1} \rightarrow S_{j+2} \rightarrow \ldots \rightarrow S_n$. In case 3, the wafer processing sequence is $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_{j-1} \rightarrow S_j \rightarrow S_{j+1} \rightarrow S_{j+2} \rightarrow \ldots \rightarrow S_n$. There is only one path for the wafer processing steps for these two cases. Furthermore, the corresponding scheduling methods for these scenarios have already been proposed in [4] and [39].

In case 2, the WFPs are $(m_1, \ldots, m_{j-1}, m_{j\&j+1}, m_{j+2}, \ldots, m_n)$ and $(m_1, \ldots, m_{j-1}, m_j, m_{j+1}, m_{j+2}, \ldots, m_n)$. This indicates that there are two processing paths: 1) $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_{j-1} \rightarrow S_{j\&j+1} \rightarrow S_{j+2} \rightarrow \ldots \rightarrow S_n$ and 2) $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_{j-1} \rightarrow S_j \rightarrow S_{j+1} \rightarrow S_{j+2} \rightarrow \ldots \rightarrow S_n$. This implies that after executing step $j-1$, the wafer can follow two possible processing routes.

Specifically, the patterns in Fig. 2 correspond to these cases. Pattern (a) corresponds to case 1, where all wafers are processed by the combined step 1 & 2. Pattern (b) corresponds to case 2, where wafers may take either the combined route S1&S2 or the sequential route S1 → S2. Patterns (c) and (d) correspond to case 3, where no wafers undergo the combined step, and all must go through S1 and S2 sequentially.

In particular, case 2 introduces additional scheduling challenges, since multiple feasible processing routes coexist. The scheduler must determine not only the wafer sequence but also the appropriate allocation between the combined and independent steps, which significantly increases complexity. The dynamic selection between alternative routes is highly combinatorial, making it difficult for traditional static scheduling methods to find optimal solutions. This complexity motivates the adoption of adaptive decision-making strategies, such as reinforcement learning, which can learn optimal or near-optimal scheduling policies by interacting with the system and dynamically handling multiple feasible processing routes.

However, no existing studies have specifically addressed the scheduling problem of DACTs with MPMs. This work aims to utilize a reinforcement learning approach, as detailed later, to schedule the entire process, covering its start-up to close-down stages.

In a semiconductor cluster tool, the number of processing modules is typically no more than eight, and the number of MPMs ranges from three to six, which means the total number of feasible configurations remains limited (e.g., fewer than 16 in our case); therefore, the computational complexity of Algorithm 1 is polynomial.

IV. MD3QN Scheduling Algorithm

In this section, we first define the environment of a DACT with MPMs for reinforcement learning. Then, we extend the D3QN algorithm by incorporating mask techniques and a prioritized replay buffer, which significantly enhances the efficiency of the exploration process.

A. Environment Definition

In reinforcement learning, a Markov decision process (MDP) can model dynamic systems with inherent randomness and decision-making characteristics from real-world problems. It is defined by the tuple $\langle S, A, P, R\rangle$, where S represents the state space, encompassing all possible states of the system; A denotes the action space, defining the set of actions triggered by each state; $P(s'|s, a)$ is the state transition probability, describing the probability distribution of transitioning to the next state s' when action a is taken in state s; and $R(s, a)$ is the reward function, which evaluates the immediate reward obtained by executing action a in state s.

In an MDP, the current state s is sufficient to fully characterize the system's dynamics, making future states dependent only on the current state and the action taken, independent of the past states. The goal of reinforcement learning for an MDP is to determine an optimal policy π^* that maximizes the expected cumulative discounted reward $G = E\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$, where γ is the discount factor balancing the importance of short-term and long-term rewards.

When scheduling DACT with MPMs, the decision-making process must be based on the current state of each PM and its associated wafer to select the most appropriate robot action.

We utilize the MDP to construct the reinforcement learning environment for scheduling a DACT.

For the deep reinforcement learning environment, the valid patterns generated by Algorithm 1 are used to configure the wafer processing steps associated with PMs.

1) *Actions:* Action Swap i with $i \in N_q^+$ refers to the wafer exchange operation between the PMs in step i. Swap i of the robot involves unloading the processed wafer from a PM in step i, followed by rotating two arms and loading another wafer into the PM for processing. Specifically, when the robot executes swap operations for PMs in step 1, it must move to LLs to unload a raw wafer and then move to a PM in step 1. This process ensures that the wafer is ready to be loaded into the PM as soon as the unloading operation ends.
2) *States:* A state is composed of the following key parameters.
   a) W_{LL}: Number of raw wafers in the LL.
   b) S_{stg}: Operation stage (start-up, steady-state, close-down).
   c) P_{Arm}: Positions of the two robot arms.
   d) N_{PM}: The number of wafers being processed.
   e) T_{RP}: The remaining processing time of a wafer in a PM.
   f) T_{RR}: The allowable wafer residency time in a PM.
   g) T_{Idle}: Idle time of PM.

   The dimensionality of the state space directly influences both the efficiency and effectiveness of reinforcement learning. While high-dimensional states can provide more detailed information, enabling the agent to better understand the environment, they can also lead to the curse of dimensionality, which increases complexity and slows learning.
3) *Reward:* The reward is given according to the observation of the cluster tool system.

Upon completing a batch of wafers, a reward of +1000 is granted to emphasize the primary objective of successful scheduling. Conversely, a substantial penalty of −1000 is applied to the violations of the basic operation rules, which is imposed on violations of basic operation rules to avoid unsafe behaviors or wafer loss required strictly in practical industrial environments. A small reward of 30 indicates a successful action. The effective utilization of parallel PMs leads to a reward of five; otherwise, underutilization incurs a reward of −5. Since the goal is to minimize the makespan, a penalty is applied to the execution time of each action.

The reward magnitudes are empirically selected to balance task completion, safety constraints, and efficiency objectives, and the learning process mainly depends on relative reward differences rather than absolute values.

The specific process is determined by the operation rules, which identify the robot actions. First, one arm unloads the processed wafer from a PM; then, the other arm rotates and loads a new wafer into the same PM. This wafer is the one that has been unloaded during Swap i–1. At this point, the entire swap operation is completed. When $i = 1$, the action involves loading the wafer from the LL into the first PM to initiate processing.

We have the following rules to guide the robot actions.

1) If the arm is holding a wafer, it cannot execute the unloading operation; conversely, if the arm does not hold a wafer, it cannot execute the loading operation.
2) Each PM can process only one wafer at a time.
3) Wafers must strictly follow the predefined sequence of processing steps.

By these rules, effective collaboration between the PM, LLs, and the robot provides a solid foundation for the deep reinforcement learning algorithm to optimize the makespan.

B. MD3QN Algorithm

Deep reinforcement learning combines the strengths of deep learning and reinforcement learning, effectively integrating the perceptual capabilities of deep learning with the decision-making capabilities of reinforcement learning [40], [41], [42]. Traditional Q-learning is a foundational algorithm in reinforcement learning that finds optimal strategies by iteratively updating a Q value table. The Bellman equation iteratively updates Q-values based on observed transitions and rewards. The update rule is given as

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'\right) - Q(s_t, a_t)\right] \quad (1)$$

where α is the learning rate, γ is the discount factor, r_{t+1} is the immediate reward, and $\max_{a'} Q(s_{t+1}, a')$ represents the estimated value of the best action in the next state. By repeatedly applying this update rule, the Q-learning algorithm converges to the optimal Q-values, enabling the agent to make optimal decisions.

Instead of maintaining a Q-value table, DQN uses a deep neural network as a function approximator to estimate the Q-values, effectively bridging the gap between reinforcement learning and deep learning.

To stabilize training and improve performance, DQN incorporates two key mechanisms. First, the experience replay is employed, where transitions (s_t, a_t, r_t, s_{t+1}) are stored in a replay buffer. During training, mini-batches are randomly sampled from this buffer, reducing correlation between updates and ensuring more stable convergence.

Second, a target network is introduced to provide a stable reference for Q-value estimation. This network, updated periodically to match the primary Q-network, mitigates instability caused by rapidly changing Q-values during the learning process. These techniques allow DQN to handle complex, high-dimensional tasks effectively. The Q-value update in DQN is defined as

$$Q(s_t, a_t; \theta) = r_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'; \theta^-\right) \quad (2)$$

where θ represents the weights of the Q-network, and θ^- represents the weights of the target network. By periodically synchronizing θ^- with θ, the target network provides a more stable training signal.

D3QN extends DQN by integrating two key innovations: double Q-learning and the dueling network architecture. These techniques address limitations in Q-value estimation, improving learning stability and performance in complex environments [43], [44].

Double Q-learning mitigates the overestimation bias inherent in standard Q-learning by decoupling the action selection and value evaluation processes. Specifically, one Q-network selects the action that maximizes the Q-value, while the other network evaluates the Q-value of that action. This adjustment results in more accurate value estimation, particularly in noisy or stochastic environments. The target value in double Q-Learning is calculated below

$$y_j = r_j + \left(1 - \ done_j\right) \times \gamma \times Q_{\theta^-}\left(s^{'}, \arg\max_{a'} Q_\theta\left(s^{'}, a^{'}\right)\right). \tag{3}$$

Here, r_j represents the immediate reward, and $(1-\ \text{done}_j)$ ensures that no future rewards are considered when the state s' is terminal ($\text{done}_j = 1$). The discount factor γ balances the importance of immediate and future rewards. The action a' is selected using the current Q-network Q_θ, while the target Q-network Q_θ^- evaluates the Q-value of this action in the next state s'. This decoupling of selection and evaluation reduces overestimation and stabilizes the learning process.

The dueling network architecture further enhances the Q-value representation by decomposing it into two separate components: the state value function $V(s)$, which estimates the overall value of being in a state, and the advantage function $A(s, a)$, which measures the relative importance of each action in that state. The Q value is then computed as follows:

$$Q(s, a; \theta) = V(s; \theta) + A(s, a; \theta) - \frac{1}{|A|}\sum_{a'} A\left(s, a^{'}; \theta\right). \tag{4}$$

Here, $(1/|A|)\sum_{a'} A(s, a^{'}; \theta)$ ensures that the advantage function is normalized to maintain numerical stability. By explicitly modeling the value and advantage separately, the dueling network architecture enhances learning efficiency, particularly in scenarios where many actions have similar Q-values.

Due to the limitations of traditional experience replay buffers, which sample uniformly and randomly without considering the varying importance of different experiences, they often suffer from inefficient sample utilization and slow convergence in practical applications. To overcome these shortcomings, we adopt a PER buffer (PERB), which assigns priorities to experiences based on their significance, enabling more frequent sampling of critical experiences to enhance learning efficiency and convergence performance.

PERB evaluates the importance of each experience using the temporal-difference (TD) error, defined by

$$\delta = \left| r_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a^{'}\right) - Q\left(s_t, a_t\right)\right|. \tag{5}$$

Experiences with larger TD errors are critical for improving the policy and are thus assigned higher sampling priorities. The sampling probability of an experience i is given by

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \quad p_i = \delta_i + \epsilon \tag{6}$$

where α controls the degree to which priorities affect sampling, and ϵ is a small constant to prevent zero priorities. This mechanism ensures that important experiences are sampled more frequently to accelerate learning.

To mitigate potential biases caused by over-reliance on high-priority samples, importance sampling (IS) weights are introduced to adjust the gradient updates

$$w_i = \left(\frac{1}{N \times P(i)}\right)^\beta \tag{7}$$

where N is the replay buffer size, and β is a parameter that adjusts the degree of correction, typically annealed from a small value to one over time to balance exploration and exploitation.

Furthermore, we incorporate a masking technique into the D3QN framework to enhance the efficiency of exploration. This technique imposes constraints on the action selection process, allowing the agent to explore only within the valid action space.

Specifically, at each state, invalid actions are masked to restrict the agent's decision space. For example, actions such as unloading a wafer from an idle PM or loading a wafer into an occupied PM with a wafer are prohibited. By effectively reducing the exploration space, the masking technique prevents unnecessary exploration of infeasible actions, significantly improving the agent's exploration efficiency.

By integrating the aforementioned mask technique and PERB, our proposed MD3QN algorithm exhibits significant advantages in exploration efficiency, sample utilization, and policy optimization. These enhancements enable it to converge more rapidly to high-quality policies while ensuring a more stable training process. The MD3QN network architecture is shown in Fig. 3.

C. Applying MD3QN Algorithm and Comparison

We illustrate the application of Algorithm 2 using the valid pattern <1, 1, 1> from the previously mentioned $k = 3$ and $l = 4$ settings.

For the pattern <1, 1, 1>, the WFP is given by WFP = (1 ⊕ (1, 1), 2, 2). We input WFP and the corresponding processing parameters into the environment to complete the initialization with a batch of 25 wafers. Following initialization, the MD3QN algorithm begins training, with the agent selecting actions within the constrained action space $A_{valid}(s_t)$, determined by masking invalid actions based on the current system state. An epsilon-greedy strategy is applied to balance exploration and exploitation, where actions are either randomly chosen from $A_{valid}(s_t)$ or selected based on the maximum Q-value estimated by the current Q-network Q_θ. The agent learns to minimize the makespan and maximize resource utilization through interaction with the environment, while the experience replay buffer and target network stabilize the learning process.

To validate the effectiveness of introducing the mask and PERB techniques, we conduct ablation experiments by comparing their performance with the basic D3QN algorithm during the training process. As shown in Fig. 4, we present

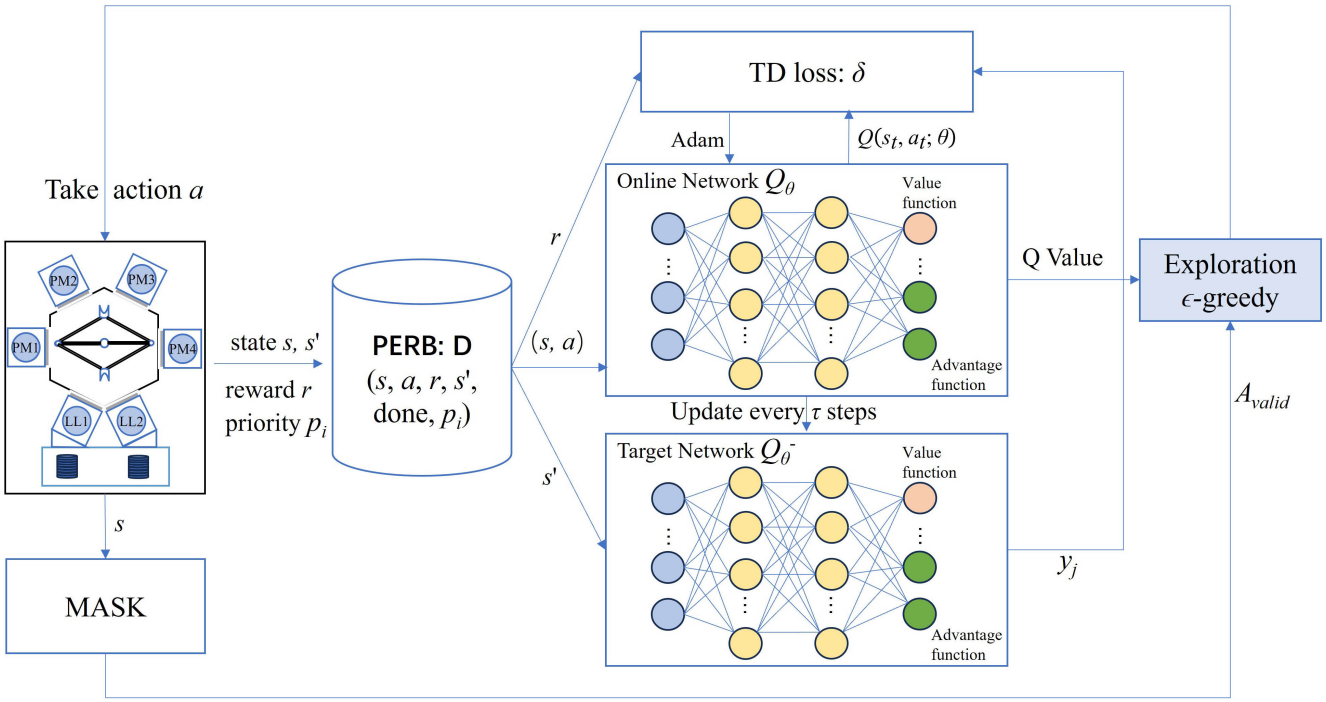

Fig. 3. MD3QN network architecture.

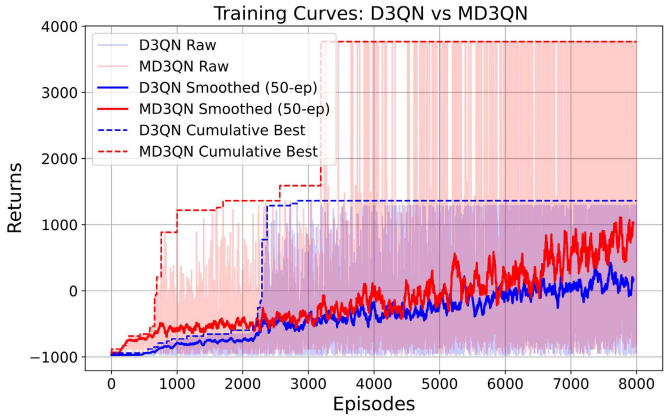

Fig. 4. Variation of episode rewards for D3QN and MD3QN algorithm.

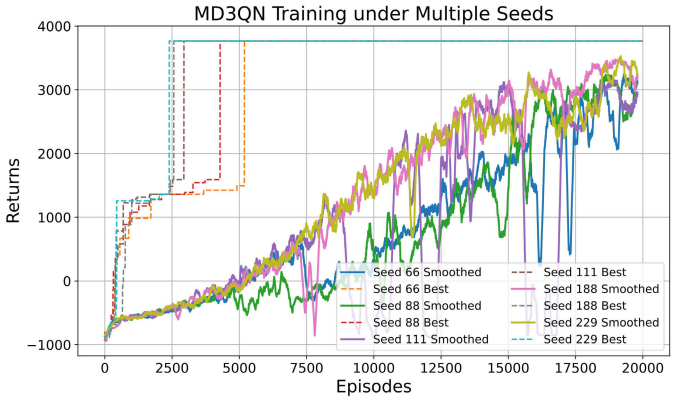

Fig. 5. MD3QN training performance across multiple seeds.

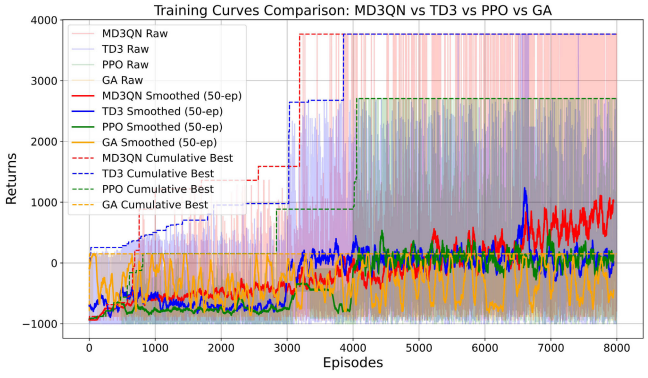

Fig. 6. Comparison of learning curves across different algorithms.

the training curves of both algorithms, including raw rewards, smoothed rewards (50-episode moving average), and cumulative best rewards. For the standard D3QN, the cumulative best rewards plateau around 1000, indicating that after over 2000 episodes, the algorithm fails to improve further. The smoothed rewards gradually increase but eventually stabilize at a suboptimal level, indicating that D3QN is unable to fully process a batch of wafers. This suggests a limited exploration capability in high-dimensional sparse reward environments, making it difficult for the agent to escape local optima and effectively transition between steady-state and close-down stages.

In contrast, the MD3QN algorithm significantly improves performance. After over 3000 episodes, returns already approach 3767, as reflected in the cumulative best rewards, indicating the emergence of optimal strategies. Although some fluctuations remain, the smoothed rewards show an upward trend, with the curve still increasing at 8000 episodes, sug-

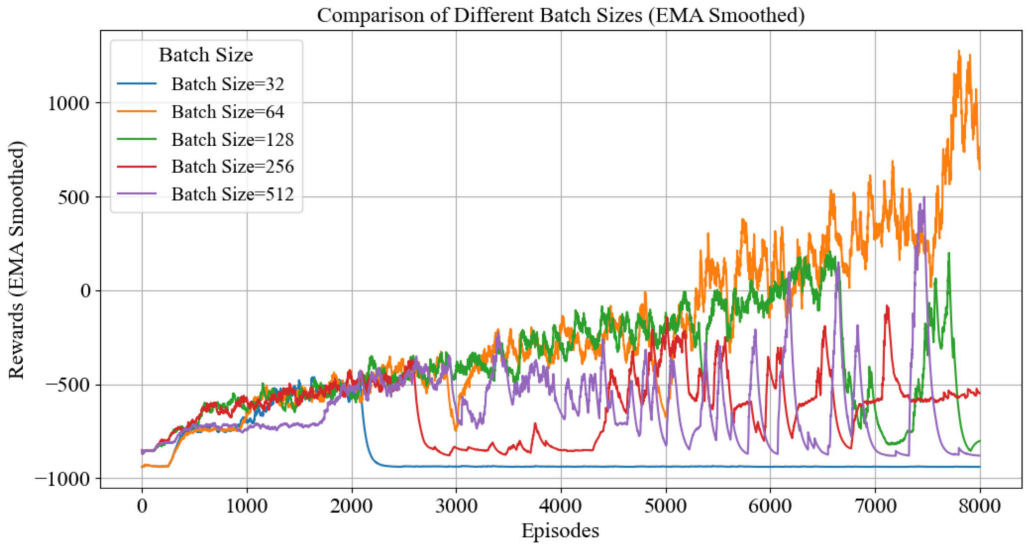

Fig. 7. Batch size sensitivity analysis.

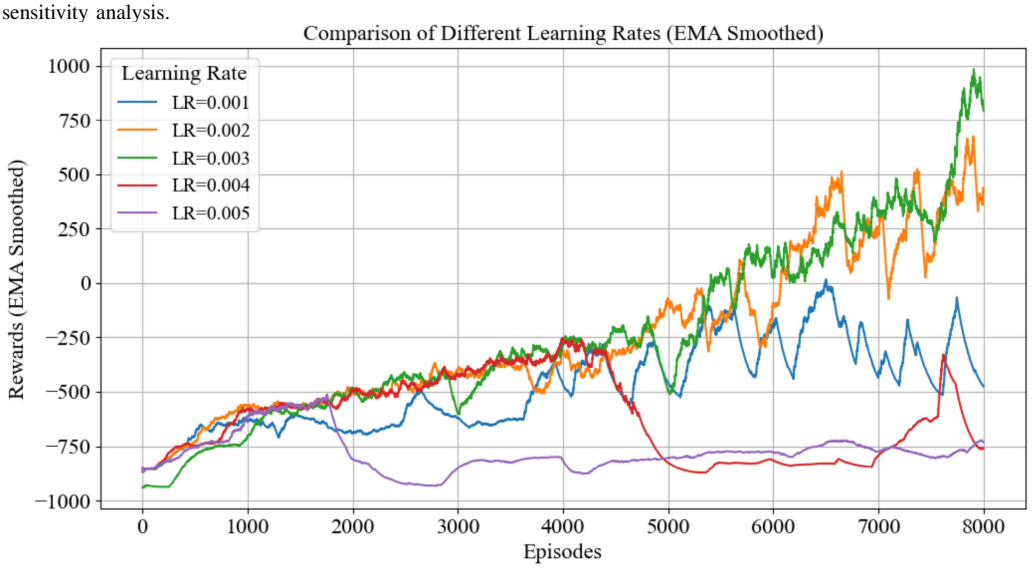

Fig. 8. Learning rate sensitivity analysis.

gesting that the agent is steadily converging toward globally optimal strategies. The subsequent multiseed comparison, as shown in Fig. 5 further demonstrates that MD3QN can maintain stable convergence within 20 000 episodes, highlighting its robustness and superior exploration efficiency. Specifically, we test five different random seeds. Across all seeds, the cumulative best rewards quickly reach the maximum return of 3767 within 2500–5000 episodes. The smoothed rewards exhibit a consistent upward trend, indicating gradual improvement of the learned policy. Although certain runs experience sudden drops, the performance typically recovers within a few hundred episodes and continues following the prior growth trajectory. This phenomenon reflects the algorithm's ability to overcome occasional suboptimal exploration or unstable policy updates, ultimately converging to a stable optimal strategy. By around 20 000 episodes, the smoothed curves across all seeds approach convergence, confirming that MD3QN not only accelerates the finding of optimal solutions but also achieves reliable stability under different initializations.

As shown in Fig. 6, MD3QN consistently outperforms baseline methods in terms of convergence stability and solution quality under the same training budget. Since hyperparameters play a critical role in determining the stability and convergence behavior of deep reinforcement learning algorithms, it is essential to analyze their influence on MD3QN. Furthermore, we conduct a hyperparameter sensitivity analysis to gain deeper insights into the performance of the MD3QN under different settings. As shown in Figs. 7–9, we analyze the impact of learning rate (lr), batch size, and gamma (γ) on the algorithm's performance. The results indicate that some parameters have clear optimal ranges. For example, $\gamma = 0.91$, 0.94, 0.95, and 0.98 demonstrate relatively stable performance, with $\gamma = 0.91$ achieving the best balance between convergence speed and stability. Similarly, learning rates $lr = 0.002$ and 0.003 provide both faster convergence and higher final rewards. In terms of batch size, 64 is identified as the optimal setting, striking a good balance between solution quality and training efficiency.

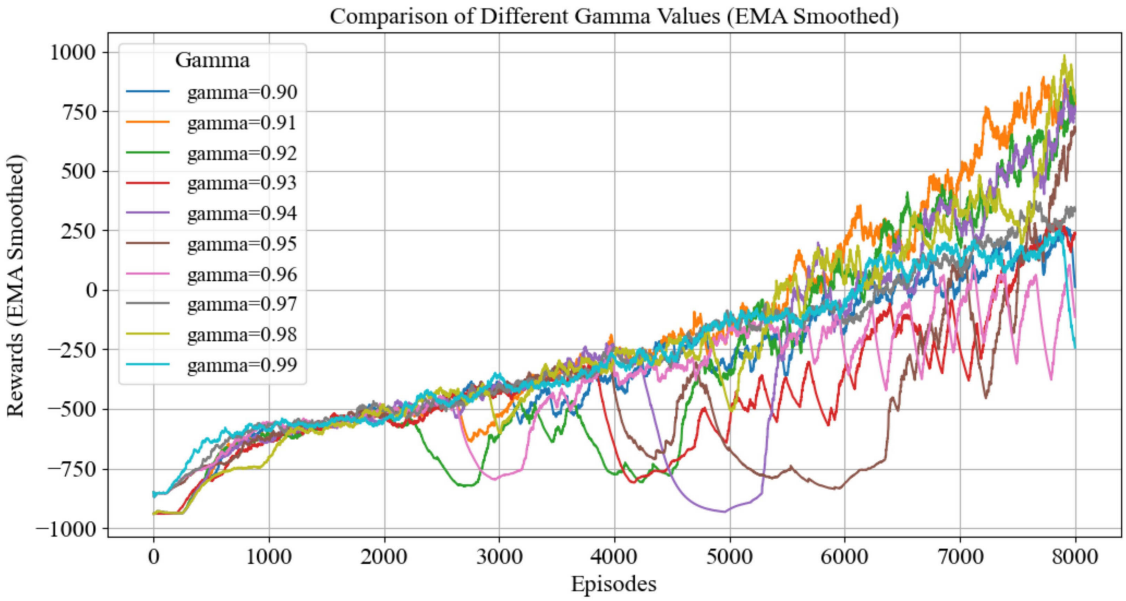

Fig. 9. Gamma sensitivity analysis.

Algorithm 2 MD3QN

1. **Input**: initial parameters:
   online network Q_θ and target network Q_θ^- // D3QN
   priority α, ϵ and importance-sampling β. // PERB
   lr, γ, ϵ_{start}, ϵ_{decay}, B, M, M_{min}, τ, // Training
2. **Output**: Trained Q_θ.
3. **Initialize** Q_θ and $Q_\theta^- \leftarrow Q_\theta$, replay buffer D and $step = 0$.
4. **For** episode = 1 to e **do**:
   $s \leftarrow$ Reset environment, $done \leftarrow$ False.
   While not $done$:
   1) step $\leftarrow$ step +1;
   2) $A_{valid} \leftarrow$ Mask(s);
   3) **If** random() $< \epsilon$:
      $a \leftarrow$ Random action from A_{valid}, $\epsilon' = \epsilon \times \epsilon_{decay}$
   4) **Else**:
      $a \leftarrow \text{argmax}\{a' \in A_{valid}\}\ Q_\theta(s, a')$, $\epsilon' = \epsilon \times \epsilon_{decay}$
   5) s', r, $done \leftarrow$ Env.step(a)
   6) Calculate δ by (5) where $a' \in A'_{valid}$.
   7) Calculate priority p_i by (6).
   8) Store (s, a, r, s', done, p_i) into D.
   9) $s \leftarrow s'$.
      If size(D) > $M_{\min}$:
   10) Sample batch from D using priority p_i^α.
   11) Calculate w_i by (7).
   12) For each sample: Calculate y_j by (3), where $a' \in A'_{valid}$.
   13) Perform gradient descent to update θ.
   14) Update priority in D: $p'_j = \left|y_j - Q_\theta\left(s_j, a_j\right)\right| + \epsilon$.
   15) **If** $step\ \%\ \tau == 0$:
      $Q_{\theta^-} \leftarrow Q_\theta$
      End If
   End If
5. **End For**

This analysis not only helps identify an optimal hyperparameter combination, as presented in Table I but also demonstrates the reliability and efficiency of the method in both theoretical and practical scenarios. These findings enhance the algorithm's applicability in dynamic environments and highlight its potential for industrial deployment.

TABLE I
D3QN PARAMETER SETTING

Parameter	Value	Parameter	Value
Learn rate: lr	0.003	Initial epsilon: ϵ_{start}	0.1
Discount factor: γ	0.91	Epsilon decay: ϵ_{decay}	0.99
Target update frequency: τ	10	Replay buffer size: M	100000
Batch size: B	64	Minimum buffer size: M_{min}	500

To further evaluate the performance and generalizability of MD3QN, we compare MD3QN with PPO, TD3, and a genetic algorithm (GA) within 8000 training episodes to assess their relative convergence behavior, exploration efficiency, and solution quality.

For a fair comparison, the GA-generated scheduling solutions are evaluated within the same simulation environment as the *RL* agents, using the identical reward function to compute their returns for the corresponding action decisions. Consequently, we ensure that all algorithms are assessed under consistent performance criteria.

As shown in Fig. 6, GA exhibits poor performance, with returns fluctuating between −1000 and 200 throughout the training process, indicating difficulty in discovering effective scheduling strategies. PPO initially achieves returns below 1000 during the first 4000 episodes and subsequently stabilizes around 3000, failing to reach the optimal solution.

TABLE II
EXPERIMENTAL RESULTS FOR EXAMPLE 1

Instance No.	Pattern	WFP	Makespan(s)	Cycle time(s)	Max residency time(s)
1	<3, 0, 0>	(3, 2, 2)	**2165**	**70.67**	77
2	<1, 1, 1>	(1 ⊕ (1,1), 2, 2)	2330	78.78	122
3	<0, 2, 1>	(2, 1, 2, 2)	3166	112	127
4	<0, 1, 2>	(1, 2, 2, 2)	2933	102	97

TD3 demonstrates intermittent success, with maximal returns appearing around 4000 episodes, but largely converges below 3000 for most of the later episodes. In contrast, MD3QN steadily discovers and maintains high-return strategies, demonstrating superior convergence speed, stability, and exploration efficiency. In Section V, we extend the comparative analysis to include mixed integer programming (MIP), providing a benchmark [45] from exact optimization methods and further validating the generalizability of MD3QN across different problem instances.

V. EXAMPLES

Example 1: A lot consisting of 25 raw wafers is processed in a DACT configured with $m_{1\oplus 2} = 3$, $m_3 = m_4 = 2$. Steps 1 and 2 are executed by MPMs, while steps 3 and 4 are handled by SPMs. The processing time for each step is given as $\alpha_1 = 90$ s, $\alpha_2 = 100$ s, $\alpha_3 = 100$ s, $\alpha_4 = 100$ s, and function switching time $\beta = 10$ s. The robot task time is $\lambda = 5$ s and $\mu = 2$ s.

By applying Algorithm 1, we identify four valid patterns, as shown in Table II. For each pattern, an environment is constructed to employ the MD3QN algorithm. The parameters of the MD3QN algorithm are shown in Table I, where the essential settings are determined based on sensitivity analysis. We use the epsilon-greedy strategy, with epsilon decaying throughout the episodes. This ensures a balance between exploration in the early stage and exploiting experiences in the late stage, accelerating the convergence while maintaining stability. In Fig. 4, the results for instance 2 demonstrate convergence within 5000 episodes.

Table II presents the results of Example 1, including all valid patterns and the corresponding WFP for each pattern. The MD3QN algorithm provides the optimized scheduling solution, yielding the makespan for each pattern under the optimal scheduling scheme. It also calculates the cycle time per wafer in the steady state and the maximum wafer residency time during the entire process.

To validate the efficiency of our experimental results, we conduct a comparison of the outcomes for each pattern. For instances 1, 3, and 4, the conditions are consistent with those in [39]. Upon verification, the cycle time derived from our solution matches the optimal cycle time reported in [39], with a detailed verifying process provided later in this section. However, it is important to note that the method in [39] is only applicable to instances 1, 3, and 4, and cannot be extended to the complex instance 2. Moreover, it focuses on optimizing the steady-state cycle time without considering transient scheduling issues. In contrast, our method is capable of solving all instances and provides a complete scheduling action sequence for the entire process, reducing wafer postprocessing residency time.

For instance 2, there is currently no research report on scheduling for this specific case. To address this, we create a mixed-integer programming (MIP) model. The efficiency of MIP has been demonstrated in [46]. Referring to the study by Bao and Wang [47], we develop the MIP model for pattern 2 and solve it using the efficient commercial solver Gurobi on a PC configured with a 13th Gen Intel Core i9-13900H CPU (2.60 GHz) and 16 GB RAM. Table III presents the results obtained by solving the MIP model for different numbers of wafers, along with the time required, and compares them with the solutions obtained using our proposed MD3QN algorithm.

The results show that for instances with fewer than eight wafers, the MIP can find solutions in a relatively short time. However, for the instance with nine wafers, the MIP takes 1638.38 s to get a solution, and for ten wafers, no solution can be found within an hour. In contrast, the MD3QN algorithm consistently provides solutions in less than one second, even for a batch of 25 wafers, with the makespan equal to the MIP solution if feasible. This demonstrates the efficiency and practicality of our proposed MD3QN algorithm.

The experimental results demonstrate that the trained MD3QN can generate an efficient schedule in less than one second, significantly improving the real-time scheduling performance. This makes the proposed method highly applicable in real production environments.

Therefore, in practical wafer processing, the optimal MPMs allocation scheme and corresponding robot scheduling action sequences can be selected based on the minimum makespan to achieve high production efficiency. At the same time, both wafer postprocessing residency time and makespan can be considered to ensure high efficiency while maintaining wafer quality. The detailed implementation is provided in Example 2.

Based on the solutions derived from the MD3QN algorithm, the action sequence for an optimal schedule in instance 2 of Example 1 is as follows: $S_1 \to S_0 \to S_1 \to S_2 \to S_1 \to S_2 \to S_3 \to S_0 \to S_4 \to S_1 \to S_2 \to S_3 \to S_5 \to S_1 \to S_2 \to S_4 \to S_5 \to \ldots \to S_0 \to S_3 \to S_6 \to S_1 \to S_2 \to S_4 \to S_5 \to S_1 \to S_2 \to S_3 \to S_6 \to S_0 \to S_4 \to S_5 \to S_1 \to S_2 \to S_3 \to S_6 \to S_1 \to S_2 \to S_4 \to S_5 \to \ldots \to S_5 \to S_2 \to S_3 \to S_6 \to S_4 \to S_5 \to S_3 \to S_6 \to S_5 \to S_6$.

TABLE III
COMPARISON RESULTS BETWEEN MIP AND MD3QN WITH WFP = (1⊕ (1, 1), 2, 2)

Instance No.	WFP= (1⊕(1,1), 2, 2) ,<α1, α2, α3, α4> = <90 s, 100 s, 100 s, 100 s>, <λ, μ> = <5 s, 2 s>, β = 10 s						
	Number of wafers	MIP				MD3QN	
		Number of Integer variables	Number of Constraints	Time taken(s)	Makespan(s)	Time taken(s)	Makespan(s)
5	1	94	69	<0.1	450	<1	450
6	2	348	217	<0.1	464	<1	464
7	3	762	445	<0.1	583	<1	583
8	4	1336	753	0.52	658	<1	685
9	5	2070	1141	1.56	737	<1	737
10	6	2964	1609	7.21	813	<1	813
11	7	4018	2157	19.78	907	<1	907
12	8	5232	2785	82.2	959	<1	959
13	9	6606	3493	1638.38	1035	<1	1035
14	10	8140	4281	>3600	-	<1	1150
15	15	18210	9421	>3600	-	<1	1517
16	25	50350	25701	>3600	-	<1	2330

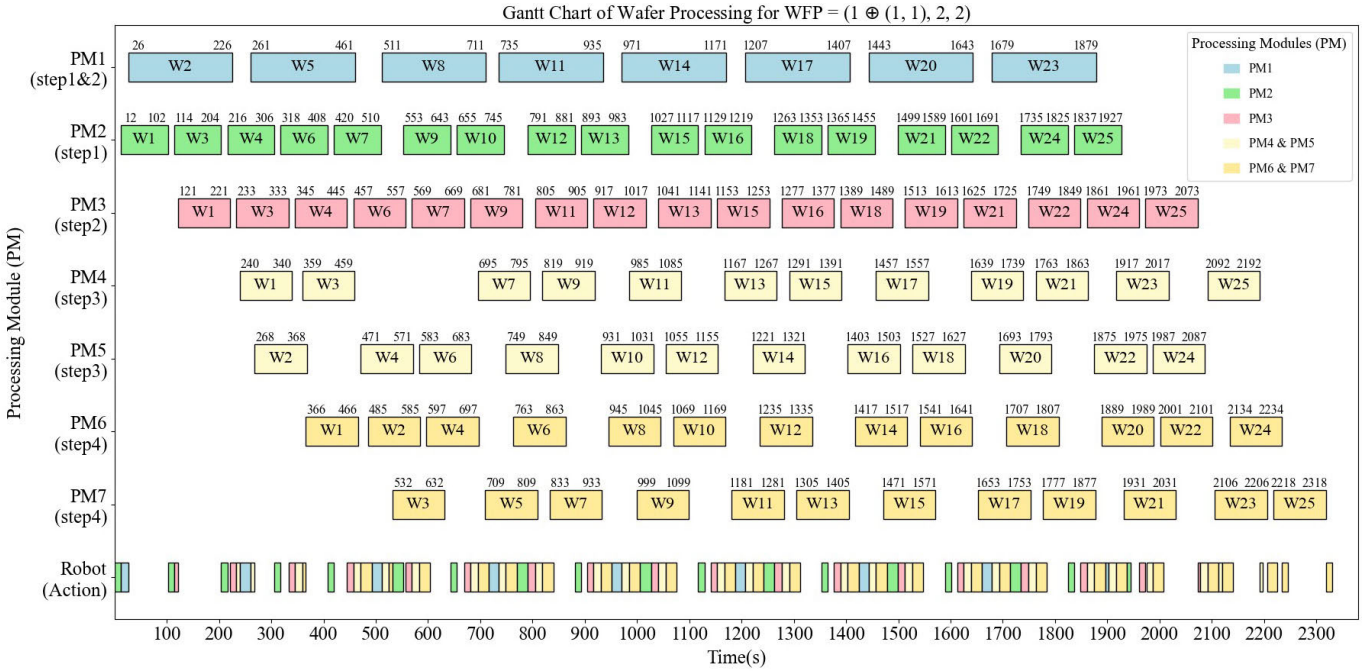

Fig. 10. Gantt chart for instance 2 in Example 1.

The corresponding Gantt chart for the action sequence is given in Fig. 10.

The process for verifying and solving 1, 3, and 4, based on the method in [39], is as follows: for instance 1, the parameters are given as follows: $m_1 = 3$, $m_2 = 2$, $m_3 = 2$, $\alpha_1 = 200$ s, $\alpha_2 = \alpha_3 = 100$ s, $\delta_1 = 200$ s, $\delta_2 = \delta_3 = 100$ s, $\lambda = 5$ s, $c = 12$ s, and $\mu = 2$ s. We have $\Pi_{1L} = 70.67$ s, $\Pi_{1U} = 137.77$ s, $\Pi_{2L} = 56$ s, $\Pi_{2U} = 106$ s, $\Pi_{3L} = 56$ s, $\Pi_{3U} = 106$ s, and $\psi_1 = 54$ s. Thus, $\Pi = 70.67$ s. $\tau_1 = 200$ s, $\tau_2 = 129.66$ s and $\tau_3 = 129.66$. Since each step satisfies $\alpha \le \tau \le \alpha + \delta$; therefore, $\Theta = \Pi = 70.67$ s.

For instance 3, the parameters are given as follows: $m_1 = 2$, $m_2 = 1$, $m_3 = 2$, $m_4 = 2$, $\alpha_1 = 90$ s, $\alpha_2 = \alpha_3 = \alpha_4 = 100$ s, $\delta_1 = \delta_2 = 200$ s, $\delta_3 = \delta_4 = 100$ s, $\lambda = 5$ s, $c = 12$ s, and $\mu = 2$ s. We have $\Pi_{1L} = 51$ s, $\Pi_{1U} = 151$ s, $\Pi_{2L} = 112$ s, $\Pi_{2U} = 312$ s, $\Pi_{3L} = 56$ s, $\Pi_{3U} = 106$ s, $\Pi_{4L} = 56$ s, $\Pi_{4U} = 106$ s, and $\psi_1 = 68$ s. Thus, $\Pi = 112$ s. $\tau_1 = 212$ s, $\tau_2 = 100$ s and $\tau_3 = \tau_4 = 212$ s. Since each step satisfies $\alpha \le \tau \le \alpha + \delta$; therefore, $\Theta = \Pi = 112$ s.

For instance 4, the parameters are given as follows: $m_1 = 1$, $m_2 = 2$, $m_3 = 2$, $m_4 = 2$, $\alpha_1 = 90$ s, $\alpha_2 = \alpha_3 = \alpha_4 = 100$ s, $\delta_1 = \delta_2 = 200$ s, $\delta_3 = \delta_4 = 100$ s, $\lambda = 5$ s, $c = 12$ s,

TABLE IV
EXPERIMENTAL RESULTS FOR EXAMPLE 2

m1⊕2 = 4, m3 =3, m4 = 4 ,<α1, α2, α3, α4> = <135 s, 145 s, 320 s, 440 s>, <λ, μ> = <5 s, 2 s>, β = 10 s					
Instance No.	Pattern	WFP	Makespan(s)	Cycle time(s)	Max residency time(s)
17	(4, 0, 0)	(4, 3, 4)	**3824**	**113**	304
18	(0, 1, 3)	(1, 3, 3, 4)	4691	147	**142**
19	(0, 3, 1)	(3, 1, 3, 4)	5016	157	329
20	(0, 2, 2)	(2, 2, 3, 4)	3833	**113**	146
21	(1, 1, 2)	(1⊕ (1, 2), 3, 4)	3891	118	240
22	(1, 2, 1)	(1⊕ (2, 1), 3, 4)	3972	121	314
23	(2, 1, 1)	(2⊕ (1, 1), 3, 4)	**3824**	**113**	275

and μ = 2 s. We have Π_{1L} = 102 s, Π_{1U} = 302 s, Π_{2L} = 56 s, Π_{2U} = 156 s, Π_{3L} = 56 s, Π_{3U} = 106 s, Π_{4L} = 56 s, Π_{4U} = 106 s, and ψ_1 = 68 s. Thus, Π = 102 s. τ_1 = 90 s, $\tau_2 = \tau_3 = \tau_4$ = 192 s. Since each step satisfies $\alpha \le \tau \le \alpha + \delta$; therefore, $\Theta = \Pi$ = 102 s. The detailed execution sequence and corresponding Gantt chart for instances 1, 3, and 4 are provided in Appendix for further reference.

Notably, in Example 1, the optimal solution (Pattern 1) assigns all MPMs to complete step 1 & 2. To investigate whether this allocation remains efficient in other scenarios, we conduct additional experiments with the results presented in Table IV.

Example 2: A lot consisting of 25 raw wafers is processed in a DACT configured with: $m_{1\oplus 2}$ = 4, m_3 = 3, m_4 = 4. Steps 1 and 2 are executed by MPMs, while steps 3 and 4 are handled by SPMs. The processing time for each step is given as α_1 = 135 s, α_2 = 145 s, α_3 = 320 s, α_4 = 440 s, and function switching time β = 10 s. The robot task time is λ = 5 s and μ = 2 s.

Table IV shows that for instances 17 and 23, both the makespan and cycle time are the shortest, with a cycle time of 113 s per wafer for the steady state. However, instance 23 has a shorter postprocessing residency time of 275 s, compared to 304 s for instance 17. Therefore, to achieve the highest processing efficiency, the MPMs allocation pattern and corresponding robot scheduling method for instance 17 are optimal. This demonstrates that allocating all MPMs to step $j\&j+1$ is not always the optimal decision.

At the same time, instance 20 also has a steady-state cycle time of 113 s. However, $m_{1\&2}$ = 0 meaning there are no MPMs to complete step 1 & 2, which results in more wafer swap operations in the cluster tool for instance 20, and additional time spent on the start-up and close-down phases. Consequently, the makespan for instance 20 is 9 s longer than the previous two instances. Nevertheless, its postprocessing residency time is only 146 s, which is 129 s shorter than that of instance 23. This indicates that when considering both makespan and wafer postprocessing residency time, the MPMs allocation pattern for instance 20 is an excellent choice, as it significantly reduces the wafer postprocessing residency time while only adding 9 s to the makespan.

We observe that although instance 18 has the shortest maximum residency time, its makespan and cycle time are relatively large, making it suboptimal.

The MD3QN framework is designed to be generally applicable across different WFPs and valid allocation configurations, as it adopts a unified state representation, action space, and reward structure that captures the fundamental characteristics of scheduling a DACT. However, the resulting scheduling policy is inherently WFP-dependent, since different WFP and allocation patterns correspond to different process flows, timing constraints, and system dynamics.

In practice, the same MD3QN network architecture and training settings are reused across different configurations, while the policy network is trained to adapt to each specific environment. Such retraining does not require redesigning the algorithm or reward structure, and the training process remains stable and computationally feasible.

VI. CONCLUSION

This work addresses the scheduling problem of DACTs with MPMs in semiconductor manufacturing. MPMs can either execute multiple steps consecutively or perform a specific step. Allocating MPMs to steps directly impacts system efficiency, necessitating process engineers to select an optimal allocation scheme based on given conditions. We propose an allocation algorithm to determine all feasible patterns. Based on these patterns, we construct a simulation environment and propose an improved D3QN approach to schedule the cluster tool processing of a batch of wafers, generating an action sequence that minimizes the makespan. Furthermore, we also consider the maximum wafer residency time, although it is not explicitly specified, we monitor it during the MD3QN execution process

to select a feasible pattern with a shorter residency time when their makespans are identical.

Compared to traditional optimization-based methods, such as the MIP model. The MD3QN can obtain a near-optimal schedule in a very short time, which is crucial for the real-world production requirements in semiconductor manufacturing, greatly reducing idle machine time due to equipment waiting and switching.

In practical manufacturing, multicluster tools are widely used for their high efficiency. However, the scheduling of multicluster tools becomes more complex due to the coordination between buffering modules and multiple robots. In future work, we will continue to employ deep reinforcement learning to address the scheduling problem of multicluster tools with MPMs. Exploring transfer or meta-learning techniques to further improve cross-configuration generalization is also an interesting future direction.

REFERENCES

[1] T.-G. Lee, T.-S. Yu, and T.-E. Lee, "Cleaning plan optimization for dual-armed cluster tools with general chamber cleaning periods," *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 3, pp. 1890–1906, Jul. 2023.

[2] J.-H. Lee and H.-J. Kim, "The impact of processing time variations on swap sequence performance in dual-armed cluster tools," *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 4, pp. 2668–2677, Oct. 2023.

[3] W. Kim, T.-S. Yu, and T.-E. Lee, "Integrated scheduling of a dual-armed cluster tool for maximizing steady schedule patterns," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 12, pp. 7282–7294, Dec. 2021.

[4] Q. Zhu, M. Zhou, Y. Qiao, N. Wu, and Y. Hou, "Multiobjective scheduling of dual-blade robotic cells in wafer fabrication," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 12, pp. 5015–5023, Dec. 2020.

[5] T.-S. Yu and T.-E. Lee, "Wafer delay analysis and control of dual-armed cluster tools with chamber cleaning operations," *Int. J. Prod. Res.*, vol. 58, no. 2, pp. 434–447, Jan. 2020.

[6] H.-J. Kim and J.-H. Lee, "Closed-form expressions on lot completion time for dual-armed cluster tools with parallel processing modules," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 898–907, Apr. 2019.

[7] J. Wang, H. Hu, C. Pan, Y. Zhou, and L. Li, "Scheduling dual-arm cluster tools with multiple wafer types and residency time constraints," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 3, pp. 776–789, May 2020.

[8] Q. Zhu, Y. Qiao, and N. Wu, "Optimal integrated schedule of entire process of dual-blade multi-cluster tools from start-up to close-down," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 2, pp. 553–565, Mar. 2019.

[9] T.-S. Yu and T.-E. Lee, "Scheduling dual-armed cluster tools with chamber cleaning operations," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 1, pp. 218–228, Jan. 2019.

[10] Y. Qiao, N. Wu, F. Yang, M. Zhou, Q. Zhu, and T. Qu, "Robust scheduling of time-constrained dual-arm cluster tools with wafer revisiting and activity time disturbance," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 6, pp. 1228–1240, Jun. 2019.

[11] S.-G. Ko, T.-S. Yu, and T.-E. Lee, "Scheduling dual-armed cluster tools for concurrent processing of multiple wafer types with identical job flows," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 3, pp. 1058–1070, Jul. 2019.

[12] Q. Zhu, G. Zhang, Y. Hou, and N. Wu, "Scheduling a dual-arm robotic two-cluster tool in the event of a process module failure," *Expert Syst. Appl.*, vol. 297, Feb. 2026, Art. no. 129415.

[13] Y. Qiao, M. Zhou, N. Wu, Z. Li, and Q. Zhu, "Closing-down optimization for single-arm cluster tools subject to wafer residency time constraints," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 11, pp. 6792–6807, Nov. 2021.

[14] J. Wang, C. Pan, H. Hu, L. Li, and Y. Zhou, "A cyclic scheduling approach to single-arm cluster tools with multiple wafer types and residency time constraints," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 3, pp. 1373–1386, Jul. 2019.

[15] F. Yang et al., "Efficient approach to cyclic scheduling of single-arm cluster tools with chamber cleaning operations and wafer residency time constraint," *IEEE Trans. Semicond. Manuf.*, vol. 31, no. 2, pp. 196–205, May 2018.

[16] F. Yang, N. Wu, Y. Qiao, M. Zhou, and Z. Li, "Scheduling of single-arm cluster tools for an atomic layer deposition process with residency time constraints," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 3, pp. 502–516, Mar. 2017.

[17] N. Wu, C. Chu, F. Chu, and M. C. Zhou, "A Petri net method for schedulability and scheduling problems in single-arm cluster tools with wafer residency time constraints," *IEEE Trans. Semicond. Manuf.*, vol. 21, no. 2, pp. 224–237, May 2008.

[18] N. Wu and M. Zhou, "A closed-form solution for schedulability and optimal scheduling of dual-arm cluster tools with wafer residency time constraint based on steady schedule analysis," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 2, pp. 303–315, Apr. 2010.

[19] T. Nishi and I. Matsumoto, "Petri net decomposition approach to deadlock-free and non-cyclic scheduling of dual-armed cluster tools," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 1, pp. 281–294, Jan. 2015.

[20] S.-G. Ko, T.-S. Yu, and T.-E. Lee, "Wafer delay analysis and workload balancing of parallel chambers for dual-armed cluster tools with multiple wafer types," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 1516–1526, Jul. 2021.

[21] Q. Zhu et al., "Scheduling single-arm multicluster tools for two-type wafers with lower-bound cycle time," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 11, pp. 6658–6671, Nov. 2023.

[22] M. Lee, J. R. Morrison, and A. A. Kalir, "Practical queueing models for preventive maintenance plan optimization: Multiple maintenance types and numerical studies," *IEEE Trans. Semicond. Manuf.*, vol. 34, no. 1, pp. 104–114, Feb. 2021.

[23] J.-H. Lee, H.-J. Kim, and T.-E. Lee, "Scheduling cluster tools for concurrent processing of two wafer types," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 525–536, Apr. 2014.

[24] J.-H. Lee and T.-E. Lee, "Concurrent processing of multiple wafer types in a single-armed cluster tool," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Aug. 2011, pp. 102–107.

[25] Y. Qiao, M. Zhou, N. Wu, and Q. Zhu, "Scheduling and control of startup process for single-arm cluster tools with residency time constraints," *IEEE Trans. Control Syst. Technol.*, vol. 25, no. 4, pp. 1243–1256, Jul. 2017.

[26] Q. Zhu, M. Zhou, Y. Qiao, and N. Wu, "Petri net modeling and scheduling of a close-down process for time-constrained single-arm cluster tools," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 3, pp. 389–400, Mar. 2018.

[27] F. Yang, N. Wu, Y. Qiao, and R. Su, "Polynomial approach to optimal one-wafer cyclic scheduling of treelike hybrid multi-cluster tools via Petri nets," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 270–280, Jan. 2018.

[28] L. Bai, N. Wu, Z. Li, and M. Zhou, "Optimal one-wafer cyclic scheduling and buffer space configuration for single-arm multicluster tools with linear topology," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 10, pp. 1456–1467, Oct. 2016.

[29] T.-S. Yu, H.-J. Kim, and T.-E. Lee, "Scheduling single-armed cluster tools with chamber cleaning operations," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 705–716, Apr. 2018.

[30] Y. Qiao, Y. Lu, J. Li, S. Zhang, N. Wu, and B. Liu, "An efficient binary integer programming model for residency time-constrained cluster tools with chamber cleaning requirements," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 1757–1771, Jul. 2022.

[31] Q. Zhu, H. Li, C. Wang, and Y. Hou, "Scheduling a single-ARM multi-cluster tool with a condition-based cleaning operation," *IEEE/CAA J. Autom. Sinica*, vol. 10, no. 10, pp. 1965–1983, Oct. 2023.

[32] W. Xiong, J. Li, Y. Qiao, L. Bai, B. Huang, and N. Wu, "An efficient scheduling method for single-arm cluster tools with multifunctional process modules," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 6, pp. 3270–3283, Jun. 2023.

[33] J. Wang, H. Xue, Q. Yang, and C. Pan, "A novel cyclic scheduling approach to time-constrained single-arm-robot multi-cluster tools," in *Proc. IEEE 18th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2022, pp. 1628–1633.

[34] Y. Fu, M. Zhou, X. Guo, and L. Qi, "Stochastic multi-objective integrated disassembly-reprocessing-reassembly scheduling via fruit fly optimization algorithm," *J. Cleaner Prod.*, vol. 278, Jan. 2021, Art. no. 123364.

[35] F. Yang, N. Wu, Y. Qiao, and M. Zhou, "Optimal one-wafer cyclic scheduling of hybrid multirobot cluster tools with tree topology," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 2, pp. 289–298, Feb. 2018.

[36] B. Huang, M. Zhou, P. Zhang, and J. Yang, "Speedup techniques for multiobjective integer programs in designing optimal and structurally simple supervisors of AMS," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 1, pp. 77–88, Jan. 2018.

[37] C. Hong and T.-E. Lee, "Multi-agent reinforcement learning approach for scheduling cluster tools with condition based chamber cleaning operations," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Orlando, FL, USA, Dec. 2018, pp. 885–890.
[38] Z. Cao, C. Lin, M. Zhou, and R. Huang, "Scheduling semiconductor testing facility by using cuckoo search algorithm with reinforcement learning and surrogate modeling," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 825–837, Apr. 2019.
[39] J. Wang, C. Liu, M. Zhou, T. Leng, and A. Albeshri, "Optimal cyclic scheduling of wafer-residency-time-constrained dual-arm cluster tools by configuring processing modules and robot waiting time," *IEEE Trans. Semicond. Manuf.*, vol. 36, no. 2, pp. 251–259, May 2023.
[40] Y. Yang, L. Juntao, and P. Lingling, "Multi-robot path planning based on a deep reinforcement learning DQN algorithm," *CAAI Trans. Intell. Technol.*, vol. 5, no. 3, pp. 177–183, Sep. 2020.
[41] S. Guo, X. Zhang, Y. Du, Y. Zheng, and Z. Cao, "Path planning of coastal ships based on optimized DQN reward function," *J. Mar. Sci. Eng.*, vol. 9, no. 2, p. 210, Feb. 2021.
[42] B.-A. Han and J.-J. Yang, "Research on adaptive job shop scheduling problems based on dueling double DQN," *IEEE Access*, vol. 8, pp. 186474–186495, 2020.
[43] C. Xu, P. Zhang, H. Yu, and Y. Li, "D3QN-based multi-priority computation offloading for time-sensitive and interference-limited industrial wireless networks," *IEEE Trans. Veh. Technol.*, vol. 73, no. 9, pp. 13682–13693, Sep. 2024.
[44] M. Gök, "Dynamic path planning via dueling double deep Q-network (D3QN) with prioritized experience replay," *Appl. Soft Comput.*, vol. 158, Jun. 2024, Art. no. 111503.
[45] X. Guo et al., "Modeling and optimization of multiproduct human–robot collaborative hybrid disassembly line balancing with resource sharing," *IEEE Trans. Computat. Social Syst.*, vol. 12, no. 5, pp. 2848–2863, May 2025.
[46] C. Jung and T.-E. Lee, "An efficient mixed integer programming model based on timed Petri nets for diverse complex cluster tool scheduling problems," *IEEE Trans. Semicond. Manuf.*, vol. 25, no. 2, pp. 186–199, May 2012.
[47] T. Bao and H. Wang, "Cyclic scheduling of multi-cluster tools based on mixed integer programming," *IEEE Trans. Semicond. Manuf.*, vol. 30, no. 4, pp. 515–525, Nov. 2017.

QingHua Zhu (Senior Member, IEEE) received the Ph.D. degree in industrial engineering from Guangdong University of Technology, Guangzhou, China, in 2013.

From 2014 to 2015 and from 2017 to 2018, he was a Visiting Scholar with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. He joined Guangdong University of Technology, Guangzhou, China in 2003, where he is currently an Associate Professor with the School of Computer Science and Technology. His research interests include intelligent scheduling and optimization, intelligent manufacturing, cloud computing, edge computing, and discrete event systems.

LangJin Liu received the B.Eng. degree in engineering management from Guangdong University of Technology, Guangzhou, China, in 2023, where he is currently pursuing the M.Eng. degree in computer science and technology with the School of Computer Science and Technology.

His research interests include intelligent scheduling, reinforcement learning-based methods, and optimization.

WeiXin Liang received the B.Eng. degree in computer science and technology from Guangdong University of Technology, Guangzhou, China, in 2025. He is currently pursuing the M.Eng. degree with Guangzhou Institute of Technology, Xidian University, Guangzhou, China.

His research interests include intelligent scheduling and optimization.

Yan Hou received the B.S. and M.S. degrees in computer science from Yangtze University, Jingzhou, China, in 1999 and 2002, respectively, and the Ph.D. degree in industrial engineering from Guangdong University of Technology, Guangzhou, China, in 2016.

Since 2002, she has been with the School of Computer Science and Technology with Guangdong University of Technology, where she is currently an Associate Professor. Her current research interests include production scheduling and optimization, information systems, and software engineering.