Scalable Multi-agent Reinforcement Learning for Factory-wide Dynamic Scheduling

Jaeyeon Jang[1], Diego Klabjan[2], Han Liu[3], Nital S. Patel[4], Xiuqi Li[4], Balakrishnan Ananthanarayanan[4], Husam Dauod[4], and Tzung-Han Juang[2]

[1]Department of Data Science, The Catholic University of Korea
[2]Department of Industrial Engineering and Management Sciences, Northwestern University
[3]Department of Computer Science, Northwestern University
[4]Intel Corporation

September 20, 2024

Abstract

Real-time dynamic scheduling is a crucial but notoriously challenging task in modern manufacturing processes due to its high decision complexity. Recently, reinforcement learning (RL) has been gaining attention as an impactful technique to handle this challenge. However, classical RL methods typically rely on human-made dispatching rules, which are not suitable for large-scale factory-wide scheduling. To bridge this gap, this paper applies a leader-follower multi-agent RL (MARL) concept to obtain desired coordination after decomposing the scheduling problem into a set of sub-problems that are handled by each individual agent for scalability. We further strengthen the procedure by proposing a rule-based conversion algorithm to prevent catastrophic loss of production capacity due to an agent's error. Our experimental results demonstrate that the proposed model outperforms the state-of-the-art deep RL-based scheduling models in various aspects. Additionally, the proposed model provides the most robust scheduling performance to demand changes. Overall, the proposed MARL-based scheduling model presents a promising solution to the real-time scheduling problem, with potential applications in various manufacturing industries.

1 Introduction

Scheduling in semiconductor manufacturing is an exceptionally challenging task due to several complex characteristics. These include fluctuating demand, a

variety of product types, intricate precedence constraints of operations, hundreds of machines spread across numerous workstations, a high frequency of re-entrant phenomena, frequent machine breakdowns and maintenance, and a vast number of possible routes for a job [1, 2, 3, 4]. This complicated and challenging scheduling task has been represented, at a high level of abstraction, as a flexible job shop scheduling problem (FJSSP)[5, 6] and diverse methods have been proposed to tackle the FJSSP [7, 8, 9].

With the increasing complexity and uncertainty of modern manufacturing systems, the FJSSP has garnered considerable attention in recent years [9]. This problem builds upon the conventional job shop scheduling problem by increasing flexibility and universality to address advanced manufacturing systems such as automobile assembly, chemical material processing, and semiconductor manufacturing [10, 11]. In the FJSSP, each operation can be processed on multiple compatible machines, each with a different processing time. As a result, the FJSSP is a formidable NP-hard problem [12].

Many algorithms have been developed to address the FJSSP problem [5, 7, 8, 9]. However, most of them rely on an unrealistic assumption of a static scheduling environment where all the relevant information about the production environment is known in advance, and all future events are controlled as planned. The aim of these algorithms is to create the best deterministic scheduling plan without considering any future modifications. However, in complex manufacturing systems, dynamic events such as machine breakdowns, unscheduled maintenance, job insertion, and modification of order have become increasingly common [13]. As a result, a deterministic schedule is not usually executed as planned, deteriorating the production efficiency significantly. In addition, factory scheduling involves many hard operational constraints due to the complexity and uncertainty of semiconductor manufacturing systems. Therefore, developing a real-time scheduling method capable of handling these constraints is of utmost importance. Recently, reinforcement learning (RL) has achieved huge attention in this area due to its ability to learn a good policy by trial and error in a complex environment, thereby enabling real-time decision-making [6, 14, 15, 16].

Dynamic FJSSP (DFJSSP) has been extensively studied over the past decade, with dispatching rules and metaheuristics being commonly used techniques [17]. Dispatching rules are popular among practitioners due to their simplicity and time efficiency. However, their solutions may not be of high quality since no single best rule can be applied to all situations [16]. In contrast, metaheuristics can provide higher quality solutions by breaking down the dynamic scheduling problem into a series of static sub-problems [6]. Nevertheless, metaheuristics algorithms, such as genetic algorithm, tabu search, and particle swarm optimization, may suffer from significant computational cost due to their huge search spaces [18, 19]. Thus, their practicality is questionable.

With the rapid advancement of artificial intelligence, deep RL (DRL) has emerged as an important approach in real-time scheduling due to its practicality in adapting to numerous dynamic events in real-time [20]. Despite its performance improvement over dispatching rules and metaheuristics, existing

DRL-based methods have several limitations. Firstly, most of them heavily rely on dispatching rules. Secondly, prior work does not consider hard operational constraints in dynamic environments which is crucial for a factory-wide production environment.

The factory-wide DFJSSP presents a challenge for obtaining optimal or near-optimal solutions due to its complex nature and numerous hard constraints. Even the introduction of DRL cannot address this challenge as the entire search space becomes impractical [21]. To overcome this limitation, we introduce the concept of a multi-agent RL (MARL) approach [22] that decomposes the problem into a set of operation-specific sub-problems. In this concept, achieving team goals requires effective coordination among all agents [23, 24, 25]. However, in the context of a factory-wide production environment, guiding multiple agents with commonly used shared team reward [26] is not effective due to the complex hierarchy and intricate interconnections of numerous sequential operations. Therefore, instead of a shared reward, we use operation-wise rewards that allow each agent to focus on its assigned operation, thus significantly reducing the search space. However, cooperation may not be achieved through this approach, even if each agent performs well in its assigned operation. To address this challenge, we introduce a new outer agent proposed in [27], called the leader, which coordinates the agents assigned to each operation, which we call followers, by providing an abstract goal for communication.

In a real factory production environment, converting from one product type to another requires a significant operational cost. Therefore, strict constraints are imposed to ensure that conversions are carefully executed. Since there is limited flexibility in the conversion process, even one bad decision made by a follower can have catastrophic consequences. For instance, machines can unexpectedly become idle, resulting in a significant loss of production capacity over a long period of time. However, it is impossible to guarantee that RL agents will not make any errors. Therefore, in situations where a follower's bad decision can significantly impact performance, we implement a rule-based conversion algorithm. This allows us to apply a set of predetermined rules that supersede the follower's decision-making process. In this way, we can prevent potential performance deterioration that may occur due to a follower's decision.

We benchmarked the proposed model by comparing it with the state-of-the-art DRL-based scheduling models utilizing dispatching rules in real factory environments. Experimental results show that existing DRL-based models cannot effectively learn a scheduling strategy in complex real factory environments. This reveals that a combination of predefined dispatching rules is insufficient. In contrast, our proposed model achieved significant performance improvement during training, resulting in much better scheduling strategies. Furthermore, our proposed model was found to be more robust to increasing demand, as its performance deterioration was relatively small. Specifically, compared to the existing models in the most challenging situation with high demand, the proposed model reduces tardiness by 10.4% and improves the completion rate by 31.4% on average. Our main contributions are as follows.

- An RL-based scheduling model that is not based on human-made dispatching rules is proposed.
- An MARL concept based on leader-follower that utilizes abstract goals for cooperation among followers is successfully introduced to obtain scalability for large-scale factory-wide dynamic scheduling problems.
- A rule-based conversion algorithm is developed and incorporated to prevent catastrophic loss of production capacity.

While the methodology is based on [27] herein we modify it by incorporating features specific to real-world factory environments, including scheduled maintenance and unexpected machine breakdowns. Additionally, we omit the concept of synthetic rewards present in [27] and instead developed an operation-wise rewarding strategy alongside a rule-based conversion algorithm to prevent catastrophic loss of production capacity. The remainder of this paper is organized as follows. In Section 2, related works on RL-based dynamic scheduling are overviewed. Section 3 formulates factory-wide DFJSSP, the target scheduling problem of this paper. In Section 4, we detail the proposed model, incorporating the leader-follower MARL algorithm from [27], which we integrate into our scheduling model. This approach is utilized to coordinate multiple agents in factory-wide scheduling, a problem characterized by a directed acyclic graph (DAG). In Section 5, the proposed model is thoroughly benchmarked with various baselines including several state-of-the-art models. Finally, Section 6 concludes this study by discussing some limitations and future research directions.

2 Related works

RL has proven to be successful in various fields, including robot control [28], manufacturing [29], and gaming [24] in the past few decades. As a result, many researchers in the scheduling field have become interested in this learning technique [6, 14, 15, 16]. They have adopted the concept of training adaptable agents to choose the best action (usually dispatching rules) at decision points through numerous trial and error. More specifically, dynamic scheduling problems have traditionally been approached as Markov decision processes (MDPs), which are solved using classical RL algorithms such as Q-learning and SARSA [30]. For example, Aydin and Öztemel [31] applied the Q-learning algorithm to train an agent that selects the most appropriate rule among three popular dispatching rules: the shortest processing time (SPT) rule, the cost over time rule, and the critical ratio rule. Their goal was to determine the next job in real-time, with the aim of minimizing mean tardiness. Similarly, Wei and Mingyang [32] introduced an agent trained using Q-learning to select the best rule at each decision point after designing several composite dispatching rules. In [33], Q-learning was used to select the best dispatching rule among the FIFO rule, the earliest due date rule, and the SPT rule based on the number of waiting jobs and the

estimated total lateness. Chen et al. [34] used Q-learning to produce a single composite rule by subsequently combining several dispatching rules based on a weighted aggregation method. Finally, Shahrabi et al. [35] utilized Q-learning to obtain optimal parameters for the variable neighborhood search algorithm, a heuristic method that searches for the best schedules in dynamic job shop problems with unequal job arrival times and machine failures. While conventional RL-based scheduling methods have made significant progress, the latest production systems have presented more scheduling challenges, requiring exploration of larger state spaces [36] and making the conventional RL approaches less effective.

In advanced production systems, there are almost infinite numbers of different states, making it difficult for conventional RL algorithms to obtain a good solution within a limited time budget, which raises questions about their practicality. However, with the emergence of deep learning, DRL has made breakthroughs in real-time scheduling with large and continuous state spaces by using deep neural networks (DNNs) to estimate the value function and/or policy [37, 38]. Consequently, recent developments in the field of scheduling have largely been achieved through DRL. For instance, Waschneck et al. [39] applied the concept of MARL while using a deep Q-network (DQN) so that agents can determine the best dispatching rules for a work center to maximize machine utilization. Hu et al. [40] proposed a graph convolutional network-based DQN to address route flexibility and stochastic arrivals of products in flexible manufacturing systems. Liu, Chang, and Tseng [14] introduced an actor-critic architecture while applying a parallel training method that combines asynchronous updates and deep deterministic policy gradient (DDPG). In this work, multiple agents were optimized to determine the best rule among several simple dispatching rules for each machine. Luo [6] designed six composite dispatching rules and proposed double DQN (DDQN) to determine the best rule among the designed dispatching rules at every rescheduling point to minimize total tardiness. In the following work, Luo et al. [36] extended Luo's model to a two-hierarchy deep Q-network by introducing a higher-level DDQN that determines the optimization goal. Chang et al. [15] enhanced DDQN by proposing a soft ϵ-greedy behavior policy to balance exploration and exploitation but still require the trained agent to perform the same task of choosing a dispatching rule from a set of rules. Min and Kim [16] used a deep deterministic policy gradient algorithm to dynamically tune the parameters of the apparent tardiness cost dispatching rule, which is known to yield schedules that minimize the total weighted tardiness [41].

While the existing methods have been effective for solving dynamic scheduling problems, there are weak points. Specifically, they assume that there are no restrictions on job selection and that numerous product type conversions can be executed. However, in reality, conversion is significantly limited to minimize operational costs and maximize machine utilization, while also considering operational constraints within a factory. As a result, these existing scheduling methods cannot be applied to real-world factory environments or may significantly impair scheduling performance. Thus, this study proposes a MARL based scheduling algorithm that specifically targets real-world factory environments.

3 Problem statement

The factory-wide DFJSSP assumes that jobs with the same product type have identical unit processing times for an operation. We also assume earlier arriving jobs are processed first when multiple jobs with the same product type are waiting to be processed. When a machine undergoes a change in operation mode or product type, a conversion time depending on the previous operation/product type and the new one is needed.

We describe factory-wide DFJSSP as a planning problem consisting of a period of N shift. Given a product type p, we denote K_p to be the number of lots of demand and J_p to be the number of required operations. To produce a product p, we need to sequentially conduct a sequence of operations $\{O_{p,j}|j = 1, 2, \cdots, J_p\}$ where $O_{p,j}$ is the j-th operation of p. We can perform each operation $O_{p,j}$ on a set of compatible machines $\boldsymbol{M}_{p,j} \subseteq \boldsymbol{M}$, where $\boldsymbol{M}$ is the set of all machines. In general, most machines have the ability to produce multiple type of products and perform multiple operations. However, changing operations or product types requires conversion time. To this end, we denote CO_{p_0,o_0,p_1,o_1} to be the time required to convert from operation o_0 for product type p_0 to operation o_1 for product type p_1. To better reflect a real-world factory environment, we also impose a conversion time threshold TH on all machines for each shift. This is necessary since excessive conversion time can have negative impacts on machine utilization and productivity. Specifically, in a given shift of length S, the total conversion time for each machine cannot exceed TH. The time required to process the k-th lot of product type p on operation $O_{p,j}$ is the product of the number of units in the lot $U_{p,k}$ and the unit processing time $PR_{p,j}$.

In real-world factory environments, both scheduled maintenance to prevent machine breakdowns and unscheduled maintenance due to unexpected breakdowns must be considered. For machine l, $SM_{l,t}$ is defined as 1 if l is under scheduled maintenance at time t, and 0 otherwise. Additionally, $\boldsymbol{\Omega}_t$ represents the set of machines undergoing maintenance due to unexpected breakdowns, which is really a random event. For simplicity, we assume that unscheduled maintenance begins only after the current lot is processed.

Given the provided setting, we list the decision variables in factory-wide DFJSSP.

- $X_{p,j,k,l,t} = \begin{cases} 1, & \text{if } O_{p,j} \text{ is assigned on } l \in \boldsymbol{M}_{p,j} \text{ for the } k\text{-th lot of product } p \text{ at } t \\ 0, & \text{otherwise} \end{cases}$
- $C_{p,j,k}$: Completion time of $O_{p,j}$ for the k-th lot of product type p
- $PT_{l,t}$: product type setup of $l \in \boldsymbol{M}$ at t
- $OP_{l,t}$: operation setup of $l \in \boldsymbol{M}$ at t
- $Q_{l,t} = \begin{cases} 1, & \text{if } l \in \boldsymbol{M} \text{ is not idle at } t \\ 0, & \text{otherwise} \end{cases}$

Variable $X_{p,j,k,l,t}$ determines which machine performs $O_{p,j}$ for the k-th lot of product type p, while $Q_{l,t}$ characterizes the status of M_l. Specifically, $Q_{l,t}$ is set to 1 if l is currently processing, undergoing a setup change, or undergoing scheduled or unscheduled maintenance; otherwise, it is set to 0, as detailed in (3). Whenever there is an idle machine, a decision should be promptly made for that machine. For example, if $Q_{l,t} = 0$ for an $l \in \boldsymbol{M}_{p,j}$, and $(k-1)$-th lot of p has already been processed in $O_{p,j}$, we can set $X_{p,j,k,l,t} = 1$ to initiate k-th lot of p using l for $O_{p,j}$. Conversely, if no machines are available at time t, the decision is set by fixing $X_{p,j,k,l,t} = 0$. Formally,

$$X_{p,j,k,l,t} \leq (1 - Q_{l,t})\mathbf{1}_{\boldsymbol{M}_{p,j}}(l) \sum_{l \in \boldsymbol{M}_{p,j}} \sum_{\tau=1}^{t-1} X_{p,j,k-1,l,\tau}. \tag{1}$$

Additionally, let $D_{p,k}$ be the due time of lot k of product p. Our goal is to identify decision variables that minimize the number of delayed jobs, or equivalently, maximize the completion rate, within the planning horizon of N shifts, where each shift comprises of S decision points. In this paper, for simplicity, we assume that lots within the same product type contain nearly identical unit counts. Consequently, our objective function, formalized as (2), does not account for variations in unit numbers among lots of the same product type. In the following, the expectation is with respect to unscheduled maintenance captured by random events Ω_t. Equations and constrains (1) and (3) - (11) hold for any realization of Ω_t.

$$\text{Minimize} \quad \mathbb{E}\left[\sum_{p} \sum_{k=1}^{K_p} H_{p,J_p,k}\right], \text{ where } H_{p,J_p,k} = \begin{cases} 1, & \text{if } D_{p,k} < C_{p,J_p,k} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

subject to

$$\begin{aligned} Q_{l,t} =& \mathbf{1}_{\boldsymbol{\Omega}_t}(l) SM_{l,t} \sum_{p} \sum_{k=1}^{K_p} \sum_{j=1}^{J_p} \mathbf{1}_{\boldsymbol{M}_{p,j}}(l) \Big(\sum_{\tau=t-PR_{p,j}U_{p,k}}^{t} Z_{p,j,l,\tau} X_{p,j,k,l,\tau} \\ &+ \sum_{\tau=t-(P_{p,j}U_{p,k}+CO_{PT_{l,t},OP_{l,t},p,O_{p,j}})}^{t} (1 - Z_{p,j,l,\tau}) X_{p,j,k,l,\tau} \Big) \forall l, t, \end{aligned} \tag{3}$$

$$\text{where } Z_{p,j,l,t} = \begin{cases} 1, & \text{if } PT_{l,t} = p \text{ and } OP_{l,t} = O_{p,j} \\ 0, & \text{otherwise} \end{cases},$$

$$Q_{l,t} \leq 1 \forall l, t, \tag{4}$$

$$PT_{l,t} = \begin{cases} p, & \text{if } X_{p,j,k,l,t} = 1 \\ PT_{l,t-1}, & \text{otherwise} \end{cases}, \tag{5}$$

$$OP_{l,t} = \begin{cases} O_{p,j}, & \text{if } X_{p,j,k,l,t} = 1 \\ OP_{l,t-1}, & \text{otherwise} \end{cases}, \tag{6}$$

$$C_{p,j,k} = \sum_{l \in \mathbf{M}_{p,j}} \sum_{t=1}^{NS} (Z_{p,j,l,t} X_{p,j,k,l,t} (t + PR_{p,j} U_{p,k}) + (1 - Z_{p,j,l,t}) X_{p,j,k,l,t} \\ (t + CO_{PT_{l,t}, OP_{l,t}, p, O_{p,j}} + P_{p,j} U_{p,k})) \; \forall p, j, k, \tag{7}$$

$$C_{p,j+1,k} \geq C_{p,j,k} + PR_{p,j+1} U_{p,k} + \sum_{l \in \mathbf{M}_{p,j+1}} \sum_{t=C_{p,j,k}}^{NS} \\ (1 - Z_{p,j+1,l,t}) X_{p,j+1,k,l,t} CO_{PT_{l,t}, OP_{l,t}, p, O_{p,j+1}} \; \forall p, j, k, \tag{8}$$

$$C_{p,j,k} - PR_{p,j} U_{p,k} \geq C_{p,j,k-1} - PR_{p,j} U_{p,k-1} \; \forall p, j, k, \tag{9}$$

$$\sum_{t=1}^{NS} \sum_{l \in \mathbf{M}_{p,j}} X_{p,j,k,l,t} \leq 1 \; \forall p, j, k, \tag{10}$$

$$\left(TH - \sum_{p} \sum_{k=1}^{K_p} \sum_{j=1}^{J_p} \sum_{\tau=(n-1)S+1}^{t} (1 - Z_{p,j,l,\tau}) X_{p,j,k,l,\tau} CO_{PT_{l,t}, OP_{l,t}, p, O_{p,j}} \right) \sum_{p} \sum_{k=1}^{K_p} \sum_{j=1}^{J_p} \\ \sum_{\tau=t}^{nS} (1 - Z_{p,j,l,\tau}) X_{p,j,k,l,\tau} \geq 0 \; \forall l, t \in \{(n-1)\, S + 1, \ldots, nS\}, n \in \{1,\ 2, \ldots, N\}. \tag{11}$$

Equations (3) and (4) make sure that a machine cannot process multiple jobs simultaneously. Additionally, (5) and (6) ensure that a machine only changes its setup when necessary, such as when a new product type or operation is assigned. If a setup change occurs, the machine requires both processing time and conversion time to complete the job, but only processing time is required if there is no setup change, as stipulated in constraint (7). Here, NS represents the total number of decision points occurring within an episode, which spans the planning horizon of N shifts. To ensure that jobs are executed in the proper sequence, (8) guarantees that an operation can only be executed once the previous operation for a job is completed. The first-in, first-out (FIFO) rule is applied to lots of the same product type that are waiting for the same operation, as described in constraint (9). Equation (10) guarantees that a lot can only be assigned to one machine once for an operation. If the lot requires reentry into an operation, that reentry is considered a separate operation in sequence $\{O_{p,j} | j = 1, 2, \cdots, J_p\}$. In this way, we can incorporate the reentry feature of factory scheduling into our model. Finally, constraint (11) ensures that a conversion cannot be executed if the cumulative conversion time exceeds TH within a single shift.

4 Scheduling model for factory-wide DFJSSP

In this section, we introduce a scheduling model based on MARL designed to effectively address the factory-wide DFJSSP. The proposed model features two types of agents: followers and a leader. We begin by presenting the formulation for the followers and the leader, detailing their respective state and action representations along with the reward mechanisms. Next, we describe the simulation environment used for model training and training algorithm employed to train the agents within our scheduling model. Finally, to minimize production capacity loss by preventing catastrophic mistakes by agents, we introduce a rule-based conversion algorithm.

4.1 Follower model

Our proposed model designates each follower (i.e., a lower-level agent) the responsibility of managing its own operation. As depicted in Fig. 1a, the state of each follower includes information about the state of machines, work-in-process (WIP), and demand. Specifically, the machine state elements, such as the current setup ($PT_{l,t}$ and $OP_{l,t}$) and status ($Q_{l,t}$), are encoded using one-hot encoding to represent the current product and operation for a machine l at time t. Both scheduled and unscheduled maintenance are represented identically by setting $Q_{l,t} = 1$. The cumulative conversion time for the current shift and the next available time are also tracked to predict future availability of the machine and products. The cumulative conversion time of machine l at t is calculated based on $\{PT_{l,t_0}, OP_{l,t_0}, CO_{p,o,PT_{l,t_0},OP_{l,t_0}} | \forall o, p, t_0 \leq t\}$, and the next available time is determined from $\{U_{p,k}, PR_{p,j}, X_{p,j,k,l,t_0} | \forall p, j, k, t_0 \leq t\}$. If scheduled maintenance is in progress, the next available time is set to the end of the maintenance period. Each agent maintains these machine state elements for all machines pertinent to the target operation. Additionally, WIPs awaiting the operation and both delayed and future demand lots for all available product types in the target operation are encapsulated in the state vector. These can be readily derived from $\{D_{p,k}, C_{p,j,k}, X_{p,j,k,l,t_0} | \forall j, k, l, t_0 \leq t\}$ for product type p. The underlying formulas are given in (3)-(11). Additionally, at the beginning of each shift, the leader distributes abstract goal vectors for each follower, and the abstract goal is incorporated into the follower's state to direct them towards achieving a higher reward. Details are provided later.

Product setup changes occur intermittently in real-world factories. Consequently, the followers must make crucial decisions regarding conversion for all machines assigned to the operation, as illustrated in Fig. 1b. To accomplish this, followers produce action vectors $\boldsymbol{a}^o_{n,t}$ for all machines within the target operation o at t in the n-th shift. If multiple machines belong to operation o, the follower o's action is the concatenation of actions for all member machines. That is, $\boldsymbol{a}^o_{n,t} = (\boldsymbol{a}^{o,l}_{n,t} | l \in \text{available machines for operation } o)$, where $\boldsymbol{a}^{o,l}_{n,t}$ is the action for the available machine l in operation o. Each action vector $\boldsymbol{a}^{o,l}_{n,t}$, a one-hot encoded vector, represents two factors for a machine: whether or not to convert and the next product type. In Fig. 1b, $p_{o,i}$ is i-th available product

type in operation o and np_o is the number of available product types in operation o. If the first entry is activated, p_{o1} is selected as the next product type. Otherwise, there is no change made to the product setup. Let $\psi_{l,t}^{p_{o,i},1}$ be set to 1 if $p_{o,i}$ is chosen as the next product setup for machine l at t via the activation of $(\boldsymbol{a}_{n,t}^{o,l})_{p_{o,i},1}$. Conversely, $\psi_{l,t}^{p_{o,i},0}$ should be set to 1 either if $(\boldsymbol{a}_{n,t}^{o,l})_{p_{o,i},0}$ is activated or if another product is chosen as the setup. By default, both $\psi_{l,t}^{p_{o,i},1}$ and $\psi_{l,t}^{p_{o,i},0}$ are initialized to 0. Then, for any machine l, we have

$$\psi_{l,t}^{p_{o,i},0} + \psi_{l,t}^{p_{o,i},1} = 1 \tag{12}$$

and

$$\sum_{i=1}^{np_o} \psi_{l,t}^{p_{o,i},1} \leq 1. \tag{13}$$

The next product type is only relevant when a conversion is triggered, and conversions can only occur if there is WIP of the selected product in the operation. Let us assume that $p_{o,i}$ is selected and a conversion is initiated by an action vector for machine l at time t, corresponding to operation $o = O_{p,j}$. If there is a waiting lot k, then $X_{p_{o,i},j,k,l,t+1}$ is set to 1; otherwise, it remains 0. Conversely, if no conversion is triggered but there is an upcoming lot k for the current setup $p_{o,i}$ of machine l at time t, then $X_{p_{o,i},j,k,l,t+1}$ is also set to 1.

It is not always necessary for all followers to take action since there may be a lack of available machines, as depicted in Fig. 2b. In this situation, only the followers who have access to at least one available machine are able to produce actions. The action of the follower o at t in the n-th shift, $\boldsymbol{a}_{n,t}^{o}$, is determined according to

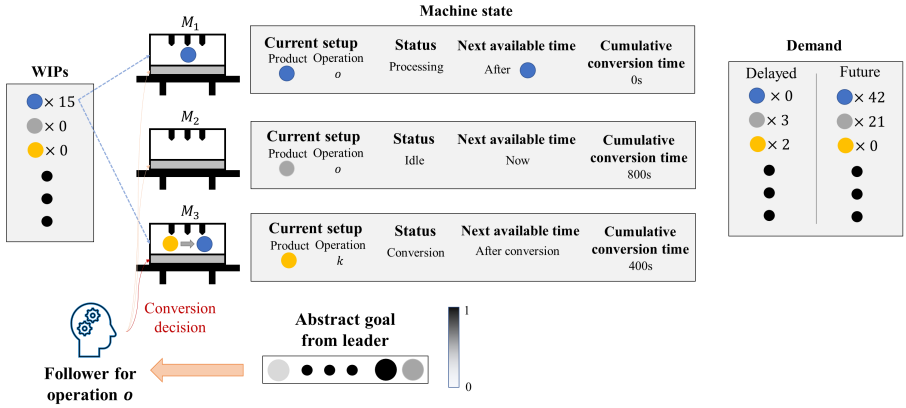
$$\boldsymbol{a}_{n,t}^{o} \sim \pi^{o}\left(\cdot \,|\boldsymbol{s}_{n,t}^{o}, \boldsymbol{g}_{n}^{o}\right), \tag{14}$$

where π^o represents the policy of the follower responsible for operation o, while $\boldsymbol{g}_n^o$ denotes the goal vector for the follower during the n-th shift.

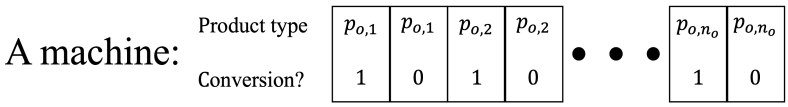
In the MARL-SR algorithm [27], synthetic rewards are utilized to account for each agent's contribution to the overall team reward. However, given that the demand for each product type is specified for each shift, we apply a more straightforward reward calculation method. At the end of each shift n, followers are penalized based on the number of delayed lots. Specifically, for the operation o, the reward r_t^o is defined as follows:

$$r_n^o = -\sum_{i=1}^{np_o} \sum_{k=1}^{K_{p_{o,i}}} H_{p_{o,i},j,k}, \text{where } H_{p_{o,i},j,k} = \begin{cases} 1, & \text{if } D_{p_{o,i},k} < C_{p_{o,i},j,k} \text{ and } D_{p_{o,i},k} \leq nS \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

This operational rewarding strategy is the default approach for our followers. The training environment is quite challenging, with the majority of actions being trivial and sparse rewards being provided. To supplement potentially insufficient training and prevent followers from generating inadequate actions that can result in catastrophic production capacity losses, we introduce a rule-based conversion algorithm in Section 4.4.



(a)



(b)

Figure 1: Illustration of (a) state and (b) action of the follower for operation o.

4.2 Leader model

Factory-wide DFJSSP requires a high level of coordination between multiple entities, such as jobs, machines, and operations. Traditional single agent based RL methods do not perform well due to the exponential growth of the joint action space as the number of entities increases [24]. To address this challenge, MARL has been proposed in which each agent aims to obtain the best policy for an entity or sub-problem [22]. This decentralization is based on the premise that achieving individual goals assigned to each agent is easier than attaining team success. However, individual success is meaningless unless team goals are achieved. Therefore, it is crucial to maximize team reward through cooperation while pursuing individual goals in MARL.

One approach of MARL is to train decentralized policies in a centralized manner, as this approach can offer valuable state information and eliminate communication barriers among agents [24, 25, 42, 43, 44]. Despite the increasing interest in centralized training with decentralized execution in the RL community, this method is not viable for use in advanced production systems due to the high synchronization demand between agents. Especially, factory-wide scheduling involves numerous precedence constraints that can be represented as a DAG. Consequently, this scheduling problem can be modeled as a multi-agent problem with DAG constraints, where each agent handles a specific operation. Essentially, every agent aims to maximize team rewards, considering the complex interrelationships between agents. A significant challenge is that rewards

can only be ascertained after multiple agents have made a series of sequential decisions over an extended period. This delay in reward distribution complicates an agent's ability to assess its contribution to team rewards at each time step, thus impeding effective learning from the perspective of the individual agent. To overcome these challenges, we use the concept of the leader proposed in [27]. This outer agent aims to coordinate the operation-specific agents towards achieving high team rewards by setting long-term, agent-specific goals.

The leader generates goals by taking into account all relevant factors, including the states of operations, and distributes them to each follower at the beginning of every shift. These goals are expressed in an abstract form as real numbers in [0, 1] and serve as communication channels between the leader and followers. The leader's rewards are based on the followers' achievements at the end of each shift. Therefore, the leader needs to create goal vectors that contain meaningful messages to lead the followers towards higher team rewards by observing the global environment information. The team rewards are also given to the leader. The role of the leader is illustrated in Fig. 2a. At the beginning of each shift, the leader receives the global state vector $\cup_o \boldsymbol{s}^o_{n,0}$, where $\boldsymbol{s}^o_{n,0}$ is the initial state vector of operation o in the n-th shift. The leader utilizes the global state vector to generate individualized goals for each follower, which are then used to address specific operations. The leader's policy π^L is defined as

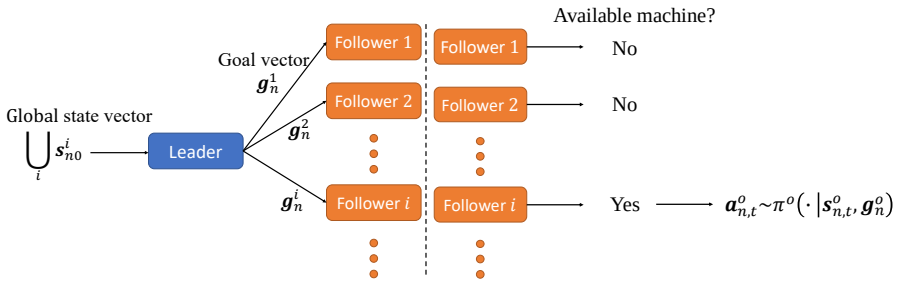
$$(\boldsymbol{g}^o_n | o \in \text{all operations}) \sim \pi^L \left(\cdot \,|\, \bigcup_o \boldsymbol{s}^o_{n,0} \right). \tag{16}$$

This approach makes it much easier to handle complex factory-wide scheduling problems, as they are decomposed into a set of weakly connected sub-scheduling problems and the problem of managing them.

While the leader has access to comprehensive information about the team's operations, followers only possess partial information. Nonetheless, they can infer the overall state of the system, including the state of the other operations and the coordination strategy of the leader, from abstract goals. Therefore, establishing effective communication channels is crucial for attaining high team rewards, utilizing goal vectors and team rewards.

The primary objective of scheduling in this study is to maximize completion rate, and the reward for maximizing completion rate can take the form of the negative value of the number of lots as introduced in Section 3. In addition, when creating schedules, it is assumed that the demand for each product type is given per shift. Thus, we calculate the penalty based on delayed lots in each operation after each shift. Then, a follower receives the sum of the penalties for all processable product types in the assigned operation as the reward after each shift based on (15). For the leader, who is responsible for overseeing the status of all operations, all delayed lots are considered when calculating the reward and it is given to the leader after each shift n as follows:

$$r^l_n = -\sum_p \sum_{k=1}^{K_p} H_{p,J_p,k}, \text{where} H_{p,J_p,k} = \begin{cases} 1, & \text{if } D_{p,k} < C_{p,J_p,k} \text{ and } D_{p,k} \le nS \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$



(a) At the beginning of n-th shift (b) At t in n-th shift

Figure 2: Overview of our MARL scheduling algorithm based on (a) leader and (b) followers. Here, π^o is the incumbent policy of the follower.

4.3 Environment and Training

In this section, we describe the simulation environment used to train RL agents, as summarized in Algorithm 1. Initially, we configure the production environment details, including the planning horizon (NS), product types (p), sequences of operations for each product type ($O_{p,j} \forall p, j \in 1, 2, \cdots, J_p$), sets of compatible machines ($\boldsymbol{M}_{p,j} \forall p, j \in 1, 2, \cdots, J_p$), unit processing times ($PR_{p,j} \forall p, j \in 1, 2, \cdots, J_p$), scheduled maintenance details ($SM_{l,t} \forall l, t$), and the conversion time threshold (TH). Each episode involves generating production plans for each shift over a predefined duration, detailing the required number of lots for each product type based on the incumbent policy. For each episode, it is assumed that demand information, specifying the required number of lots for each product type per shift, is provided. The objective of the RL agents in each episode is to meet this demand without delay. This demand information serves as the basis for reward calculation. Historical data contains only few demand observations; definitely insufficient for RL training. For this reason, we randomly generate demand for each episode with the underlying distribution calibrated from historical data, ensuring it accurately reflects the actual factory environment. We consider various factors such as the distribution of demand by product type, along with a range of operational constraints and conditions - including precedence constraints, processing times, and conversion times by product type - all based on real production datasets from a high-volume packaging and test factory. Furthermore, the initial settings of the factory environment, including the status of machines and WIP, are configured at the beginning of each episode. These initial settings lay the foundation for generating schedules based on the policies of the leader and the followers.

As is typical in RL, given an incumbent policy, at the beginning of the first shift of each episode, the leader creates and distributes goal vectors, as defined in (16), which guide the followers towards achieving high team rewards. At each decision point, machine statuses are updated to 'unavailable' based on both scheduled ($SM_{l,t} \forall l$) and unscheduled maintenance information ($\boldsymbol{\Omega}_t$), which are

not considered in the MARL-SR algorithm [27]. The distribution of the latter is generated based on historical downtimes that depend on the machine type and the specific operations performed by each machine. A decision can only be made if a machine is available, meaning it is not processing, undergoing a setup change, or undergoing scheduled or unscheduled maintenance; otherwise, no action is taken. Subsequently, the followers make conversion decisions or maintain the current product type at every decision point, including the initial time of each shift, based on (14) and a rule-based conversion algorithm (detailed in Section 4.4). Once all decisions by available followers are made, the statuses of the machines and WIP information are updated. At the end of each shift, rewards are assigned to each follower and the leader as specified in (15) and (17) respectively; however, no rewards are distributed during a shift. This is because the designated due time for meeting demand coincides with the end of the shift, resulting in a reward value of zero during the shift. As a result, after each shift, multiple (s, a, r) triplets corresponding to the number of decisions made by each follower are sampled, along with a single triplet for the leader. Each episode is stochastic with respect to the policy, demand, and unscheduled maintenance.

Although our goal is to enhance coordination through the leader, each agent functions individually, communicating via the goal vector. Consequently, samples are collected separately for each agent, as detailed in Algorithm 1. Therefore, we train each agent using their own rollouts. Specifically, we employ the proximal policy optimization (PPO) algorithm [45] as the baseline algorithm to train all agents. PPO has received huge attention due to its simplicity, robustness for a large variety of tasks, low computational complexity, and good performance [46, 47].

4.4 Rule-based conversion algorithm

Although a follower only needs to consider a single operation, there are usually numerous product candidates to choose from during a conversion. However, executing a conversion to start a product type can result in a significant loss of capacity, as it may take away the chance of producing other products due to the limited conversion time. To mitigate the risk of such a loss, we propose a rule-based conversion algorithm.

To determine the appropriate product type based on the rule-based conversion algorithm, we first score the urgency of product types for each operation using Algorithm 2. We begin by calculating the required capacity and expected remaining capacity for each product type. For a given product type p and operation o, the required capacity ($RC_{o,p}$) is the estimated total processing time for all target product WIPs in the target operation. The expected remaining capacity ($ERC_{o,p}$) is the estimated total processable time until the planning horizon ends of all machines that are currently processing p in operation o. If the required capacity is greater than the expected remaining capacity, we consider that product to be urgent and set the urgency score to the required capacity. Furthermore, a product that is not currently being processed by any machine in the operation is considered much more urgent because it cannot be produced

Algorithm 1 A summary of simulation environment

Require: Episode duration NS, all producible product types p, sequences of operations for each product type $O_{p,j}, \forall p, j \in 1, 2, \cdots, J_p$, sets of compatible machines $\boldsymbol{M}_{p,j}, \forall p, j \in 1, 2, \cdots, J_p$, unit processing times $PR_{p,j}, \forall p, j \in 1, 2, \cdots, J_p$, scheduled maintenance information $SM_{l,t} \forall l, t$, and the conversion time threshold TH

loop
  Configure the initial setup of each episode using job information ($D_{p,k}$ and $U_{p,k} \forall p, k \in 1, 2, \cdots, K_p$), as well as the initial settings for machines ($Q_{l,0}, PT_{l,0},$ and $OP_{l,0} \forall l \in \boldsymbol{M}$)
  Sample random demand
  The initial state vectors of the leader and the followers are set
  for each shift **do**
    for each decision point **do**
      Sample unscheduled maintenance
      if initial decision point **then**
        The leader creates and distributes goal vectors based on (16)
      end if
      Machine statuses are updated to account for both scheduled and unscheduled maintenance based on (3)
      All available followers take actions based on (14) and the rule-based conversion algorithm in Section 4.4
      Machine statuses are updated to account for the actions of the followers based on (3)
      if the end of the shift **then**
        Rewards are assigned to each follower and the leader as specified in (15) and (17), respectively
        The state vector of the leader is updated
      else
        Each follower receives a reward value of zero.
      end if
      The state vectors of the followers are updated
    end for
  end for
end loop

without a conversion. In this case, we add a large value (BN) to the urgency score to prioritize this product. It is important to note that BN must be greater than the maximum value of the possible required capacity. The urgency score of o and p is denoted by $US_{o,p}$

Algorithm 2 Urgency scoring

Require: Required capacity $RC_{o,p}$ and expected remaining capacity $ERC_{o,p}$ for all pairs of (operation o, product p)

Set a large value BN for urgency scoring
Initialize urgency score $US_{o,p} = 0$ for all pairs of (operation o, product p)
for each operation o **do**
  for each product type p processable in operation o **do**
    if $RC_{o,p} > ERC_{o,p}$ **then**
      if p is not being processed by any machine in operation o **then**
        $US_{o,p} = RC_{o,p} + BN$
      else
        $US_{o,p} = RC_{o,p}$
      end if
    end if
  end for
end for

After urgency scoring computation in Algorithm 2, we make a conversion decision for each available machine as outlined in Algorithm 3. We first create an action for each operation o according to policy (14). Subsequently, we decode the corresponding action for each available machine to determine two key variables: the conversion indicator (CI), which indicates whether a conversion should be triggered, and the candidate product (p_{cand}), which represents the potential next product setup. When determining the next product for an available machine m, we take into account not only the generated action for that machine but also the current product WIPs and the operation's capacity for that current product. If there are WIPs for the selected product type, we assign p_{cand} to the next product setup in the episode. Otherwise, we prioritize urgent products and select the most urgent product as the next product setup. To accomplish this, we evaluate the current product setup p_m on machine m and determine if it can be replaced with the most urgent product, $\text{argmax}_{p \in P_o} US_{o,p}$, where P_o is the set of eligible products by operation o. To ensure that there will be no missing lots of p_m, we first verify that all WIPs of p_m can be completed before the planning horizon ends. We do this by verifying that the required capacity RC_{o,p_m} is less than $ERC_{o,p_m} - ERCM_{o,m,p_m}$, the expected remaining capacity without the target machine m. Here, $ERCM_{o,m,p_m}$ refers to the expected remaining capacity of the target machine m. Additionally, we check if there is an urgent product ($\sum_{p \in P_o} US_{o,p} > 0$). If both conditions are met, we assign the most urgent product as the next product for that machine.

In summary, we introduce several key modifications to the MARL-SR algo-

Algorithm 3 Conversion decision at t in shift n

Require: Required capacity $RC_{o,p}$ and expected remaining capacity of operation $ERC_{o,p}$ for all pairs of (operation o, product p), expected remaining capacity of machine $ERCM_{o,m,p}$ for all triplets of (operation o, machine m, product p)

Urgency scoring()
for each operation o **do**
  if there is an available machine **then**
    Produce $\boldsymbol{a}^{o}_{n,t} = \{\boldsymbol{a}^{o,m}_{n,t} | m \in$ machines for operation $o\}$
    for each available machine m **do**
      Get current product setup p_m
      CI and $p_{cand} \leftarrow$ *Decode* $(\boldsymbol{a}^{o,m}_{n,t})$
      if CI=True **then**
        if there are WIPs of p_{cand} in operation o **then**
          Next product setup $p_m \leftarrow p_{cand}$
        else if $RC_{o,p_m} < ERC_{o,p_m} - ERCM_{o,m,p_m}$ and $\sum_{p \in P_o} US_{o,p} > 0$ **then**
          Next product setup $p_m \leftarrow \text{argmax}_{p \in P_o} US_{o,p}$
        else
          Do not change product setup
        end if
      end if
    end for
  end if
end for

rithm outlined in [27]:

- MARL-SR employs synthetic rewards to distinguish and reflect each agent's contribution to the overall team reward, as individual agent reward calculation is typically infeasible. In contrast, for factory scheduling, we can measure operation-wise performance, allowing us to implement an operation-specific reward calculation.
- Additionally, our approach models both scheduled and unscheduled machine maintenance, which are not considered in MARL-SR.
- Given that poor decisions by an agent can severely impact production capacity in factory scheduling, we also develop a rule-based conversion algorithm to override agent decisions if they could lead to significant production losses.

5 Experimental Evaluations

5.1 Implementation details

We implemented both the proposed model and the benchmark models using the PPO algorithm. For each agent, a fully-connected neural network, consisting of two layers with 256 units each and ReLU activation functions, is utilized for both the actor and the critic components. In our algorithm, the actor consists of the leader and the followers which all have such network architecture. We compared our proposed model with two benchmark models, which are described in detail later in this section. To determine the optimal hyperparameters for each model, we conducted both Kruskal-Wallis and ANOVA tests for reasonable parameter choices and groups corresponding to performance metrics. The results showed no significant differences across the four performance metrics-tardiness, number of changeovers, cumulative idle time, and completion rate against demand-within the parameter ranges tested: batch size (64-256), learning rate (0.0001-0.00001), discount factor (0.95-0.99), and lambda (0.9-0.95) for our proposed model and one of the benchmark models. For the other benchmark model, performance improvement was observed only when lambda was set to 0.95, with no statistical differences noted for the other hyperparameters. Consequently, we applied the hyperparameter settings listed in Table 1 to all models. Additionally, for each operation o in shift n, the dimension of the corresponding goal vector g_n^o is set to 3, as recommended in [27]. All models compared in this study are implemented using TensorFlow version 2.8 on a server equipped with an Intel i9-12900k CPU. Due to the shallow network architecture employed for both the actor and the critics, and the simulation's need for extensive sequential computations, GPU execution does not help.

In this study, we evaluate the effectiveness of our proposed method in real-world scheduling scenarios using two simulators built on different real production datasets for Intel's high volume packaging and test factory. We refer to the two production scenarios as short-term and long-term scenarios, as summarized in

Table 1: Hyperparameters

Hyperparameter	Value
Batch size	256
Learning rate	0.0001
Discount factor	0.9900
Clipping value	0.2000
Generalized advantage estimation parameter lambda	0.9500

Table 2. Here, the average out-degree in DAG refers to the mean number of outgoing arcs in the DAG, which is constructed based on the precedence constraints of operations. To assess performance in diverse environments, we further diversified each scenario into low-demand, medium-demand, and high-demand cases. Specifically, compared to low-demand cases, medium- and high-demand cases require three and five times the demand on average, respectively. In general, a more sophisticated scheduling strategy is required as demand increases.

To enhance exploration during training, the set of target product types required for production in each episode changes probabilistically, resulting in varying workloads for each agent. Additionally, to promote exploration within the PPO algorithm, we incorporate entropy regularization as outlined in [48], with the entropy coefficient of 0.01. This regularization helps maintain policy diversity by preventing premature convergence and fostering a broader exploration of the action space.

For each demand case in both short- and long-term scenarios, we train the comparison models over 10,000 episodes, with each model utilizing the policies that achieved the highest team reward during the last 100 episodes of training. These optimal models are then tested over 100 episodes. In each test episode, all comparison models begin scheduling under identical conditions, including the same demand information as well as both scheduled and unscheduled maintenance details.

In our algorithm, the followers learn policies to meet the demand over a predefined number of shifts. However, the number of shifts in the two cases in Table 2 is too long to enable effective learning within a reasonable timeframe. Therefore, we train the agents in simulations spanning four shifts (two days) and then apply the learned policies using a rolling horizon approach during inference. In the long-term scenario, training takes 5.62 hours, with inference requiring 2.68 minutes per episode. For the short-term scenario, training takes 5.92 hours, and inference takes 0.53 minutes per episode. Given the planning horizon for both scenarios, the time required for both training and inference is manageable.

In this study, we compare the proposed model with existing RL-based dynamic scheduling models. While several such models have been developed, most are not suitable for factory-wide scheduling because existing models do not consider factory-specific operational constraints in modeling or require excessive computational time. Two recent models - deep reinforcement learning for job

Table 2: Summary of two scenarios

Scenario	Short-term	Long-term
No. product types	35	20
No. operations	26	21
No. machines	159	115
Avg. out-degree in DAG	1.33	1.37
Duration	1 week (14 shifts)	3 weeks (42 shifts)

shop scheduling problems (DRL-JSSP) [14] and deep reinforcement learning for dynamic flexible job shop scheduling (DRL-DFJSS) [15] - can be adapted for factory-wide scheduling with some modifications. Both models use RL agents to select dispatching rules at each action point. DRL-JSSP employs a single agent per operation to choose one of the three rules, while DRL-DFJSS trains one agent to handle all operations and selects one of the four rules. However, as the third rule corresponding to DRL-DFJSS is not applicable in our problem, we excluded it from the analysis.

In a factory environment, the large state and action vectors necessary to manage hundreds of machines lead to significant learning and inference times. Moreover, the state representation and reward calculation methods used in DRL-JSSP and DRL-DFJSS are not directly applicable to our simulation, where the number of jobs demanded varies and the penalties for tardy jobs are unknown. Therefore, we employ the same multi-agent setting where an agent is assigned to each operation (with the concept of a leader being unique to our model), use a consistent state representation feature format (with our model incorporating goal vectors from the leader), and apply an operation-wise rewarding strategy across all models, focusing on comparing the algorithmic differences in job selection and agent coordination.

5.2 Comparison with benchmarks

In this section, we compare the proposed model with existing RL-based dynamic scheduling models. Fig. 3 illustrates the total reward per episode for our proposed model and the two comparison models trained on the low-demand case of the long-term production scenario. In the figure, the moving window method is applied, where the team rewards are averaged over a sliding window of 10,000 episodes. The averaging window is shifted by one episode at a time, indicating a step size of one episode per move. The DRL-JSSP and DRL-DFJSS models initially perform well due to their reliance on human-designed dispatching rules. However, their performance remains almost stagnant throughout training, indicating that a combination of dispatching rules is insufficient for large-scale factory-wide scheduling. Conversely, the proposed model exhibits poor performance initially but demonstrates a substantial improvement during training. This implies that the proposed model can effectively address the challenges of

large-scale factory-wide scheduling.

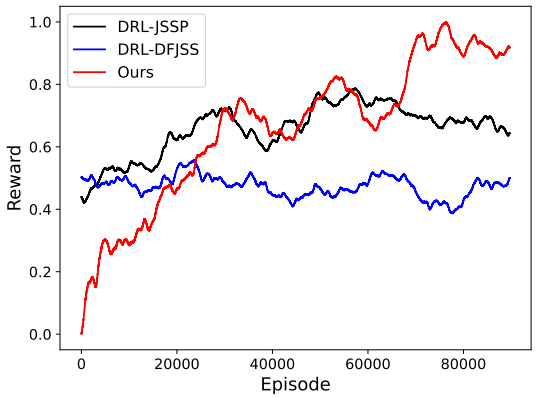


Figure 3: Training of the three models on the low-demand case of the long-term production scenario. Min-max normalization is applied to the total reward to standardize the scale of the y-axis.

Next, we utilize four metrics, namely tardiness, the number of changeovers, cumulative idle time, and completion rate against demand, to compare the models. We are unable to divulge the performance of the proposed model due to Intel's data confidentiality policy. Consequently, we establish the proposed model as the baseline and assess the improvement (percentage improvement) over benchmark models. Specifically, we report an increase in completion rate and a decrease in tardiness, the number of changeovers, and cumulative idle time as performance improvement. In summary, a negative number means that we outperform.

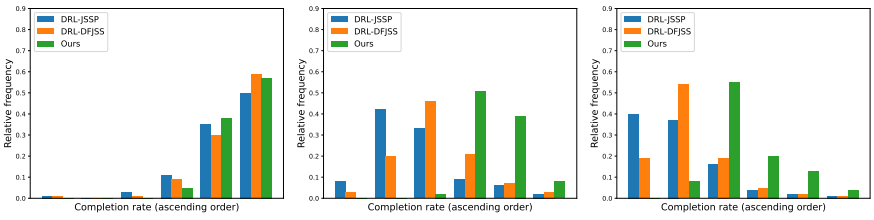
Tables 3 and 4 show comparison results for the long- and short-term production scenarios, respectively. We highlight the improvement values of the comparison models in bold if the proposed model achieves the best performance for the corresponding criterion. In summary, the proposed model achieves the best performance for 10 out of 12 criteria in both long- and short-term production scenarios. Moreover, the performance gap generally increases as the demand level rises, except in the case of the number of conversions. Notably, our proposed model consistently delivers the best performance in terms of conversions, despite this metric's apparent lack of correlation with demand levels. Although the proposed model shows a significant decline in cumulative idle time for the low-demand case of the long-term scenario, it excels in more critical metrics such as tardiness and completion rate, which are directly tied to productivity. Furthermore, despite a noticeable drop in tardiness performance during low-demand conditions in the short-term scenario, the gap is relatively minor. This is because all three models maintain very low tardiness when demand is low, as reflected by the standard deviation in percentage improvement.

We additionally compare the three models based on their completion rates

Table 3: Percentage improvement of comparison models in four metrics compared to the proposed model for the long-term production scenario

Demand	Metric	DRL-JSSP		DRL-DFJSS	
		Mean	Std.	Mean	Std.
Low	Tardiness	**-18.91**	106.65	**-10.06**	111.45
	No. conversions	**-121.50**	57.07	**-139.91**	66.26
	Cumulative idle time	17.27	7.19	16.51	7.88
	Completion rate	**-1.25**	4.51	**-0.83**	4.38
Medium	Tardiness	**-43.10**	26.72	**-27.24**	20.67
	No. conversions	0.37	34.91	-10.29	32.49
	Cumulative idle time	**-63.54**	47.56	**-53.82**	52.62
	Completion rate	**-36.35**	13.73	**-26.89**	12.41
High	Tardiness	**-12.20**	9.05	**-8.67**	8.37
	No. conversions	**-92.67**	105.09	**-111.33**	83.35
	Cumulative idle time	**-109.30**	50.27	**-116.08**	56.43
	Completion rate	**-35.10**	12.42	**-27.64**	11.71

since this metric is closely related to the number of delayed lots used in reward calculations. The results are presented in Figs. 4 and 5. The proposed model achieves a higher completion rate in most experimental settings, especially demonstrating greater proportions in the higher ranges of the histograms. This indicates a lower risk of poor performance with our model, a trend that becomes more pronounced as demand increases.



(a) Low-demand (b) Medium-demand (c) High-demand

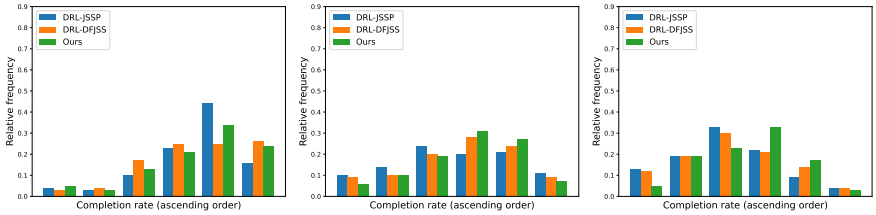
Figure 4: Completion rate histogram of three comparison models for the long-term production scenario. The x-axis tick values are omitted for confidentiality.

5.3 Ablation study

To verify the effectiveness of each component proposed in this study, we conduct a quantitative ablation study. Specifically, we generate 100 production plans based on the long-term scenario outlined in Table 2. We then compare the performance of the following models:

Table 4: Percentage improvement of comparison models in four metrics compared to the proposed model for the short-term production scenario

Demand	Metric	DRL-JSSP		DRL-DFJSS	
		Mean	Std.	Mean	Std.
Low	Tardiness	9.70	64.39	-15.32	70.83
	No. conversions	**-87.17**	73.15	**-78.72**	67.86
	Cumulative idle time	**-4.68**	11.39	**-3.51**	12.35
	Completion rate	0.24	5.04	-0.12	5.60
Medium	Tardiness	**-17.02**	52.34	**-6.19**	43.29
	No. conversions	**-78.32**	80.20	**-74.82**	73.51
	Cumulative idle time	**-11.37**	16.72	**-12.82**	16.70
	Completion rate	**-4.81**	10.99	**-1.80**	9.35
High	Tardiness	**-15.40**	29.55	**-12.36**	29.73
	No. conversions	**-59.01**	58.70	**-47.09**	50.48
	Cumulative idle time	**-21.10**	16.99	**-18.37**	18.83
	Completion rate	**-8.72**	10.41	**-5.84**	11.03



(a) Low-demand (b) Medium-demand (c) High-demand

Figure 5: Completion rate histogram of three comparison models for the short-term production scenario. The x-axis tick values are omitted for confidentiality.

1) Shared reward model (SRM): Under this model, the followers are updated using a reward-sharing strategy, where the rewards are calculated based on all delayed lots and distributed equally among all followers.

2) Operation-wise reward model (ORM): This model calculates rewards for each follower based on delayed lots of all processable products in the assigned operation.

3) Leader-follower shared reward model (LFSRM): The leader-follower concept using the abstract goal is incorporated into SRM.

4) Leader-follower operation-wise reward model (LFORM): The leader-follower concept is incorporated into ORM. Specifically, LFORM introduces operation-wise reward calculation to the LFSRM framework.

5) Proposed model: The proposed model, LFORM-RC, incorporates rule-based

conversion in addition to the LFORM approach. The acronym RC signifies the inclusion of rule-based conversion as a novel feature of the model.

We establish SRM as the baseline and assess the improvement of the other models based on the four metrics used in Section 5.2 compared to the baseline. The comparison results are presented in Table 5. It is important to note that, while a small number of conversions is desirable as it can reduce operating costs, other metrics such as tardiness and completion rate take priority, as they are crucial for achieving high productivity. The results demonstrate that the operation-wise rewarding strategy, the leader-follower concept using abstract goals, and the rule-based conversion contribute to overall performance improvement. In particular, the rule-based conversion is crucial in improving tardiness and completion rates by preventing significant capacity losses through the minimization of poor conversions.

Table 5: Percentage improvement in four metrics compared to SRM after conducting 100 simulations and averaging the results

	ORM		LFSRM		LFORM		LFORM-RC (Ours)	
	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
Tardiness	25.20	50.86	-19.81	117.53	-6.85	63.98	**84.08**	15.42
No. conversions	-0.48	37.03	11.49	24.49	11.22	24.87	-265.56	106.54
Cumulative idle time	4.27	10.32	6.47	13.26	3.01	13.16	**11.08**	8.92
Completion rate	14.24	23.48	8.11	22.46	7.45	22.99	**33.23**	23.80

6 Conclusion

In this study, we present a novel scalable MARL model for dynamic scheduling problems in real-world factories. The proposed model addresses several hurdles faced in factory scheduling by combining operation-specific agent modeling, abstract goal-based leader-follower coordination, and a rule-based conversion algorithm, which are novel in the field of RL-based scheduling. In particular, we found that RL agents can make errors that lead to significant losses in production capacity. To mitigate this issue, a rule-based conversion algorithm can be used to supplement the agents' operational decisions. Our experiments, conducted on simulators constructed using Intel production datasets, demonstrate that the proposed model significantly outperforms state-of-the-art RL-based scheduling models that rely on human-made dispatching rules. This highlights the limitations of existing methods that use a combination of human-made rules and the potential of our approach to provide more efficient scheduling solutions for real-world factories.

While our proposed model exhibits good performance and scalability, there is an issue that needs to be addressed in our future research. The model assumes that production-related information, such as operation list, machine list, and product types, remain fixed. However, in real-world factories, these settings can change from time to time, necessitating a retraining of the model. Although this

can be easily addressed with sufficient computing resources, the process requires frequent manual interventions, which can be a challenge. To overcome this, we need to develop a dynamically evolving model that can adapt to factory setting changes. This direction will be a focus of our future research efforts.

References

[1] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, "Development of dynamic scheduling in semiconductor manufacturing using a Q-learning approach," *International Journal of Computer Integrated Manufacturing*, pp. 1–17, 2021.

[2] D. K. Kim, H. J. Kim, and T. E. Lee, "Optimal scheduling for sequentially connected cluster tools with dual-armed robots and a single input and output module," *International Journal of Production Research*, vol. 55, no. 11, pp. 3092–3109, 2017.

[3] Q. Yu, H. Yang, K. Y. Lin, and L. Li, "A self-organized approach for scheduling semiconductor manufacturing systems," *Journal of Intelligent Manufacturing*, vol. 32, no. 3, pp. 689–706, 2021.

[4] Y. Chen, L.-H. Su, Y.-C. Tsai, S. Huang, and F.-D. Chou, "Scheduling jobs on a single machine with dirt cleaning consideration to minimize total completion time," *IEEE Access*, vol. 7, pp. 22 290–22 300, 2019.

[5] K. Z. Gao, P. N. Suganthan, T. J. Chua, C. S. Chong, T. X. Cai, and Q. K. Pan, "A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion," *Expert Systems with Applications*, vol. 42, no. 21, pp. 7652–7663, 2015.

[6] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Applied Soft Computing*, vol. 91, p. 106208, 2020.

[7] P. Fattahi, M. Saidi Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 18, no. 3, pp. 331–342, 2007.

[8] M. K. Amjad, S. I. Butt, R. Kousar, R. Ahmad, M. H. Agha, Z. Faping, N. Anjum, and U. Asgher, "Recent research trends in genetic algorithm based flexible job shop scheduling problems," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–32, 2018.

[9] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 904–916, 2019.

[10] X. Guo, S. Liu, M. Zhou, and G. Tian, "Disassembly sequence optimization for large-scale products with multiresource constraints using scatter search and petri nets," *IEEE Transactions on Cybernetics*, vol. 46, pp. 2435–2446, 2016.

[11] K. Gao, F. Yang, M. Zhou, Q. Pan, and P. N. Suganthan, "Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm," *IEEE Transactions on Cybernetics*, vol. 49, pp. 1944–1955, 2019.

[12] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 32, pp. 1–13, 2002.

[13] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling*, vol. 12, pp. 417–431, 2009.

[14] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, vol. 8, pp. 71 752–71 762, 2020.

[15] J. Chang, D. Yu, Y. Hu, W. He, and H. Yu, "Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival," *Processes*, vol. 10, p. 760, 2022.

[16] B. Min and C. O. Kim, "State-dependent parameter tuning of the apparent tardiness cost dispatching rule using deep reinforcement learning," *IEEE Access*, vol. 10, pp. 20 187–20 198, 2022.

[17] P. Lou, Q. Liu, Z. Zhou, H. Wang, and S. X. Sun, "Multi-agent-based proactive–reactive scheduling for a job shop," *The International Journal of Advanced Manufacturing Technology*, vol. 59, pp. 311–324, 2012.

[18] N. Kundakcı and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Computers & Industrial Engineering*, vol. 96, pp. 31–51, 2016.

[19] T. Ning, M. Huang, X. Liang, and H. Jin, "A novel dynamic scheduling strategy for solving flexible job-shop problems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 7, pp. 721–729, 2016.

[20] B. Cunha, A. M. Madureira, B. Fonseca, D. Coelho, A. M. Madureira, A. Abraham, N. Gandhi, and M. L. Varela, *Deep reinforcement learning as a job shop scheduling solver: A literature review*, ser. Hybrid Intelligent Systems. Springer, 2020, vol. 923.

[21] K. Subramanian, C. L. Isbell Jr, and A. L. Thomaz, "Exploration from demonstration for interactive reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*, 2016.

[22] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," in *Handbook of Reinforcement Learning and Control*, 2021, pp. 321–384.

[23] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI Conference on Artificial Intelligence*, 2018.

[24] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning." in *International Conference on Machine Learning*, 2018, pp. 4295–4304.

[25] J. Yang, A. Nakhaei, D. Isele, K. Fujimura, and H. Zha, "CM3: Cooperative multi-goal multi-stage multi-agent reinforcement learning," in *International Conference on Learning Representations*, 2020.

[26] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *International Conference on Machine Learning*, 1993, pp. 330–337.

[27] J. Jang, D. Klabjan, H. Liu, N. S. Patel, X. Li, B. Ananthanarayanan, H. Dauod, and T.-H. Juang, "Learning multiple coordinated agents under directed acyclic graph constraints," *arXiv preprint arXiv:2307.07529*, 2023.

[28] Y. Cui, T. Matsubara, and K. Sugimoto, "Pneumatic artificial muscle-driven robot control using local update reinforcement learning," *Advanced Robotics*, vol. 31, no. 8, pp. 397–412, 2017.

[29] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T. A. Runkler, and V. Sterzing, "A benchmark environment motivated by industrial control problems," in *IEEE Symposium Series on Computational Intelligence*, 2018, pp. 1–8.

[30] R. S. Sutton and A. G. Barto, *Reinforcement learning*. MIT press, 2018.

[31] M. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robotics and Autonomous Systems*, vol. 33, no. 2-3, pp. 169–178, 2000.

[32] Y. Z. Wei and M. Y. Zhao, "A reinforcement learning-based approach to dynamic job-shop scheduling," *Acta Automatica Sinica*, vol. 31, no. 5, pp. 765–771, 2005.

[33] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73–82, 2005.

[34] Xili Chen, XinChang Hao, Hao Wen Lin, and Tomohiro Murata, "Rule driven multi objective dynamic scheduling by data envelopment analysis and reinforcement learning," in *IEEE International Conference on Automation and Logistics*. IEEE, 2010, pp. 396–401.

[35] J. Shahrabi, M. A. Adibi, and M. Mahootchi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Computers & Industrial Engineering*, vol. 110, pp. 75–82, 2017.

[36] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Computers & Industrial Engineering*, vol. 159, p. 107489, 2021.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[38] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016, pp. 1–14.

[39] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Optimization of global production scheduling with deep reinforcement learning," *Procedia CIRP*, vol. 72, pp. 1264–1269, 2018.

[40] L. Hu, Z. Liu, W. Hu, Y. Wang, J. Tan, and F. Wu, "Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network," *Journal of Manufacturing Systems*, vol. 55, pp. 1–14, 2020.

[41] A. P. J. Vepsalainen and T. E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management Science*, vol. 33, pp. 1035–1047, 1987.

[42] F. A. Oliehoek and N. Vlassis, "Q-value functions for decentralized pomdps," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems.*, 2007, pp. 838–845.

[43] L. Kraemer and B. Banerjee, "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, vol. 190, pp. 82–94, 2016.

[44] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6382–6393.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[46] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization," in *Proceedings of International Conference on Unmanned Systems*, 2019, pp. 523–533.

[47] Z. Zhang, X. Luo, T. Liu, S. Xie, J. Wang, W. Wang, Y. Li, and Y. Peng, "Proximal policy optimization with mixed distributed training," in *International Conference on Tools with Artificial Intelligence,*, 2019, pp. 1452–1456.

[48] J. Schulman, P. Abbeel, and X. Chen, "Equivalence between policy gradients and soft Q-learning," *arXiv:1704.06440*, 2017.