IJECBE (2026), **4**, **1**, 50–68
Received (4 March 2026) / Revised (31 May 2026)
Accepted (3 June 2026) / Published (30 April 2026)
https://doi.org/10.62146/ijecbe.v4i1.227
https://ijecbe.ui.ac.id
ISSN 3026-5258

International Journal of Electrical, Computer and Biomedical Engineering

RESEARCH ARTICLE

Energy Consumption Optimization for Flexible Job-Shop Scheduling in Manufacturing Industry Using Multi-Agent Reinforcement Learning

Satwika Bintang Bahana and Naufan Raharya*

Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia, Depok, Indonesia
*Corresponding author. Email: naufanraharya@ui.ac.id

Abstract

Modern manufacturing industries are increasingly demanding production systems that are both flexible and efficient in handling dynamic operational conditions. Traditional scheduling approaches, such as the job-shop scheduling Problem (JSSP), are limited by the constraint that each operation must be executed on a predefined machine. To address this limitation, the flexible job-shop scheduling Problem (FJSSP) was introduced, allowing alternative machine options for each operation and thereby enhancing system flexibility. This study proposes a scheduling optimization approach based on Multi-Agent Reinforcement Learning (MARL), to support decision-making in complex production environments. Experimental results demonstrate that the proposed method reduces energy consumption by up to 40.62% compared to the heuristic method. Moreover, the model shows superior performance in controlling the worst-case makespan and achieves significantly higher success rates in satisfying various production constraints compared to all tested heuristic and metaheuristic methods.

Keywords: flexible job-shop scheduling problem (FJSSP), energy optimization, Multi-Agent Reinforcement Learning (MARL)

1. Introduction

Scheduling in a manufacturing industry is the process of allocating and sequencing production operations to limited resources in order to improve operational efficiency and maximize system performance [1, 2, 3]. It can be widely classified into two categories, which are flow shop and job shop. In the flow-shop scheduling problem,

every job follows an identical processing route through all machines arranged in series, where each stage corresponds to a specific machine and all jobs must be processed in the same order [4]. Despite its relevance, flow-shop scheduling lacks flexibility because it assumes that all jobs share the same machine sequence, limiting its capacity in heterogeneous production settings. Here, the heterogeneous production setting means that there are various operation types and machine capabilities. In contrast, the job-shop scheduling problem (JSSP) allows each job to be defined as a sequence of operations that must be processed on specific machines in a fixed order [5]. We can generalize the JSSP into a flexible job shop scheduling problem (FJSSP) by allowing each operation to be processed on alternative machines, which introduces additional complexity by requiring simultaneous routing (machine selection) and sequencing (operation order decisions) [1]. As the number of jobs, operations, and machines increases, the computational complexity grows rapidly, making large-scale instances extremely challenging to solve optimally. Robust scheduling is therefore essential to maintain efficient and adaptive production in dynamic environments.

The FJSSP is NP-hard, and as the problem size increases it becomes unrealistic to obtain an optimal schedule for every instance within a reasonable time [5, 6, 7, 8, 9]. Exact approaches, including mathematical programming and constraint programming, can return optimal solutions when sufficient computation time is available; otherwise, the resulting best-effort solutions exhibit optimality gaps that typically widen as the instance scale grows [10, 11, 12, 13]. Heuristic methods, in particular priority dispatching rules, are popular because they are simple to implement, require little computation, and are often used to quickly build initial schedules [14]. However, such rules provide no optimality guarantees, and their schedules can be far from the optimum [12, 8, 9, 10]. Metaheuristics are commonly adopted to obtain near-optimal solutions within limited time; nevertheless, they do not guarantee global optimality and may require long computation times under certain conditions [8, 10, 14].

Recent studies have shown that reinforcement learning (RL) offers significant potential for solving the flexible job-shop scheduling problem (FJSSP) and its variants, particularly in dynamic and uncertain manufacturing environments. In this context, recent approaches have leveraged graph-based representations combined with deep reinforcement learning (DRL) to effectively capture job–machine relations and minimize makespan [7, 15]. Nevertheless, single-agent DRL frameworks tend to centralize decision-making and face rapidly expanding action spaces when jointly sequencing operations and assigning machines, which limits scalability on large or highly dynamic shop floors. To improve scalability, a hierarchical RL framework was proposed that splits large-scale FJSSP into a higher-level rescheduling agent and two lower-level agents for operation sequencing and machine assignment, while the higher-level control remains centralized [8]. However, these studies primarily focus on minimizing time-related objectives, such as makespan or tardiness, without considering energy consumption. Incorporating energy consumption into job-shop scheduling is essential, as manufacturing enterprises face growing economic and environmental pressures from high energy usage [16], and rising electricity prices combined with sustainability demands make energy costs a critical factor in production planning [17]. Recent studies on energy-aware FJSSP has used metaheuristic [16], exact approaches with

heuristic [17], and metaheuristic to minimize energy cost with makespan [18], while learning-based hybrids combine neural networks for energy estimation with genetic algorithms (GA) for scheduling [19]. However, these centralized and computationally intensive methods remain limited in scalability and adaptability.

To resolve the computational intensive in the centralized method, a decentralized Multi-Agent reinforcement learning (MARL) framework in [20, 21, 22]. Here, decentralized MARL does provide scalability and robustness, as the system is not dependent on a single controller and can continue operating if an individual agent fails. However, the policy learned by the current decentralized MARL struggles to satisfy strict global action constraints. With only local information, agents in a fully decentralized system can find it intractable to satisfy these constraints through self-decision alone, and this can also help generalization of the learned policies. While priority dispatching rules cannot provide an optimal policy on their own, they can be effectively integrated into a decentralized MARL framework to allow agents to learn more precise policies during the early stages of training. This integration avoids the lengthy and computationally expensive exploration process often required to navigate massive decision spaces. Specifically, by incorporating these rules into the action decoding or filtering logic, the system prevents agents from attempting actions that would violate the physical or logical constraints of the workshop as shown in [20, 22]. This mechanism ensures that the learning process remains focused on feasible and high quality scheduling schemes.

This study introduces a MARL framework for scheduling in a manufacturing Industry. While the foundational manufacturing environment is inspired by the research of Johnson et al. [20], which focuses primarily on minimization of makespan, the main objective of this paper is minimization of total energy consumption in *units*. In this framework, makespan is treated as a restrictive performance constraint rather than a primary optimization goal for the completion of dynamically arriving heterogeneous assembly jobs. To address these objectives, we implement a Multi-Agent Shared Deep Q-Network architecture. The contributions of this work, achieved through the development and evaluation of this Multi-Agent system, are as follows:

1. We formulate the problem with an objective that prioritizes minimization of energy consumption joint with makespan as a performance requirement, ensuring that the system optimizes energy efficiency without sacrificing timely completion of assembly tasks.
2. We propose a decentralized MARL framework that incorporates priority dispatching rules, enabling efficient, real-time, and scalable coordination of robots under partial observability, thereby overcoming the limitations of centralized approaches.
3. We evaluate and compare the proposed MARL framework against traditional heuristic and metaheuristic methods, demonstrating its superior adaptability and energy efficiency in dynamic manufacturing scenarios involving heterogeneous jobs and strict constraints.

The rest of this paper is organised as follows. Section 2 presents the poblem formulation, including the physical description of the manufacturing system, the

mathematical optimization model for energy consumption, and the definition of the Markov Decision Process (MDP) components. Section 3 details the proposed methodology, focusing on, and the action masking mechanism used for coordination. Section 4 discusses the experimental setup, provides a comparative analysis of different reinforcement learning architectures, and evaluates the performance of the proposed model against heuristic and metaheuristic benchmarks under various operational constraints. Section 5 concludes the paper by summarizing the key findings and the implications of the MARL framework for manufacturing.

2. Problem Formulation

This section establishes the formal framework for the FJSSP. We provide a detailed description of the conveyor-based manufacturing environment and formulate a mathematical optimization model focused on energy minimization. Subsequently, the scheduling task is mapped into a Markov Decision Process (MDP) to define the state, action, and reward structures for the robotic agents.

2.1 Model System Description

As shown in Figure 1, the manufacturing environment features a set of robots that process a dynamic stream of arriving jobs within a conveyor production system. Each job is drawn from the product set $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_{n_y}\}$, representing distinct product types with prescribed operation sequences. When the conveyor reaches its capacity limit, newly arriving jobs are temporarily stored in a buffer and released according to a first-in–first-out (FIFO) rule as soon as the first conveyor segment (C_1) becomes available. Jobs enter the system at C_1 and circulate following the conveyor's rotation. Robot M_r is fixed adjacent to a particular conveyor segment C_{γ_r}, from which it may pick a job, move it to the workbench for processing, and afterwards either return it to the conveyor if further operations remain or dispatch it to the completion area once all operations are finished. The highlighted window size w specifies how many adjacent segments a robot can observe locally. In this study, we set the observation range to $w = 2$, allowing robot M^m can observe its two nearest segments.

2.2 Optimization mathematical model

The primary objective of this study is to minimize the total energy consumption of all machines involved in the production process within the FJSSP system. The objective function is formulated in Equation (1) as follows:

$$\min_{p_t^m, q_t^m} \quad E_{\text{total}} = \sum_{m \in M} \sum_{t=0}^{T} \left(\eta_{\text{p}}\, p_t^m + \eta_{\text{q}}\, q_t^m \right) \tag{1}$$

Table 1. Notation Summary

Variable	Description
Constraint and Objective Variables	
p_t^m	1 if machine m is idle at time t
q_t^m	1 if machine m is working at time t
$\delta_{j,t}$	1 if job j is on the conveyor at time t
$S_{j,k}$	Start time of processing sequence k for job j
$\tau_{j,o}^m$	Actual duration of operation o for job j by machine m
T_{makespan}	Total production duration (makespan)
E_{total}	Total energy consumption over the time horizon
η_p	Energy consumption when machine is idle
η_q	Energy consumption when machine is working
$\mathcal{P}$	Power consumption unit per time
$\mathcal{C}_j$	1 if job j is completed, 0 otherwise
$\mathcal{N}_j$	Total target number of jobs to be completed
Indices	
m	Index of machine
t	index of discrete time
j	index of job
o	index of operation type
Constants	
n_m	Number of machines in the sytems
n_y	Number of product variants.
n_o	Number of unique operation types.
n_c	Number of conveyor segments.
n_{cap}	Maximum conveyor capacity (percentage).
n_{j}	Number of jobs
$T_{\max}$	Maximum production time limit
$E_{\max}$	Maximum energy consumption limit
$\tau_{j,o}$	Base processing time of an operation in job j
v^m	represents the processing speed of machine m
ω	window size
Sets	
M	Set of all machines in the system, $M = \{M^1, M^2, \ldots, \mathcal{M}^{n_{\text{m}}}\}$
$\mathcal{J}$	Set of all jobs, $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_{n_{\text{j}}}\}$
$\mathcal{O}$	Set of operation types, $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{n_{\text{o}}}\}$
$\mathcal{Y}$	Set of products, $\mathcal{Y} = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_{n_{\text{y}}}\}$.
$\mathcal{C}$	Set of conveyor segments, $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{n_{\text{c}}}\}$.

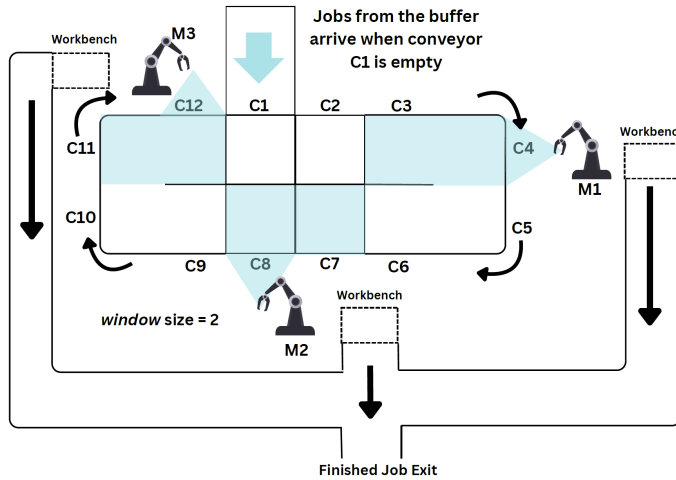

Figure 1. Manufacturing Illustration

$$\sum_{j \in J} \delta_{j,t} \leq \lfloor n_{\text{cap}}\, n_c \rfloor, \quad \forall\, t = 0, \ldots, T \tag{2}$$

$$S_{j,k} \geq S_{j,k-1} + \tau^{m}_{j,k}, \quad \forall j \in J,\ k = 2, \ldots, n_j \tag{3}$$

$$T_{\text{makespan}} \leq T_{\max} \tag{4}$$

$$\frac{E_{\text{total}}}{T_{\text{makespan}}} \leq \mathcal{P} \tag{5}$$

$$p^{m}_{t} + q^{m}_{t} = 1, \quad \forall\, m \in M,\ t = 0, \ldots, T \tag{6}$$

$$\sum_{j \in J} \mathcal{C}_j = \mathcal{N}_J \tag{7}$$

The constraints (2) to (7) of the proposed model are described as follows:

1. **Conveyor Capacity Constraint** (2): This constraint ensures that the number of jobs present on the conveyor at any time t does not exceed the maximum allowable limit. The purpose of this constraint is to prevent job congestion which could obstruct the production flow.
2. **Operation Sequence and Processing Time Constraint** (3): This constraint guarantees that the kth operation of job j can only begin after the completion of its preceding operation. The processing duration is computed based on Equation (8).
3. **Production Makespan Constraint** (4): This constraint limits the overall production duration such that it does not exceed the predefined maximum makespan $T_{\max}$.

4. **Energy Consumption per Time Interval Constraint** (5): This constraint regulates the estimated energy consumption across all machines at each time interval during the production time. The average consumption must remain below the specified threshold $\mathcal{P}$.
5. **Single Machine Status per Time Constraint** (6): This constraint ensures that at any given time t machine m can only be in one status: either idle ($p_t^m = 1$) or working ($q_t^m = 1$). A machine cannot simultaneously exist in both states or remain undefined.
6. **Job Completion Constraint** (7): This constraint guarantees that the total number of completed jobs equals the planned target number of jobs ensuring that no job is left unfinished.

The actual processing time of an operation sequence in a job is determined by the base processing time and the processing speed of the assigned machine as formulated in Equation (8):

$$\tau_{j,o}^m = \left\lceil \frac{\tau_{j,o}}{v_o^m} \right\rceil \tag{8}$$

where:

- $\tau_{j,o}$ denotes the base processing time of the o-th operation of job j,
- v_m represents the processing speed of machine m for the given operation,

2.3 MDP Formulation

The Markov Decision Process (MDP) is the fundamental framework through which an agent learns from its interactions with the environment. Based on the current state, the agent selects an action with the objective of maximizing the long-term cumulative reward. The MDP components adopted in this study are inspired by Johnson *et al.* [20]. These components are defined as follows:

1. **State (S)**: The observable state for each agent is represented as:

$$s_t^m = \left[\ell^m,\ \mathcal{O}^m,\ O_t^m,\ \mathcal{G}_t, \lambda_{\ell^m,t},\ \lambda_{\ell^m-1,t},\ \ldots,\ \lambda_{\ell^m-\omega+1,t}\right]. \tag{9}$$

   where the observation consists of the stationary position of machine ℓ^m, the set of operation types executable by the machine $\mathcal{O}^m$, the current operation type being executed $\mathcal{O}_t^m$ (with $\mathcal{O}_t^m = 0$ if machine doesn't do any operation), the status vector of all machines $\mathcal{G}_t$, and the sequence of job operations observable within the conveyor window section of size ω, denoted by $\lambda_{\ell^m-\omega+1,t}$.
   The machine status vector $\mathcal{G}_t$ represents the operational condition of each machine at time t, obtained through inter-machine communication. The classification of agent statuses is presented in Table 2.

Table 2. Agent Status Classification

Status	Description
0	Agent is in an idle state (not assigned to any operation)
1	Agent has accepted an operation from a job
2	Agent is currently executing the assigned operation
3	Agent has completed the assigned operation

2. **Action ($\mathcal{A}$)**: The action space for each robot agent is defined as a discrete set:

$$\mathcal{A} = [\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{w-1}, \mathcal{A}_d, \mathcal{A}_c], \tag{10}$$

where each action corresponds to a specific decision available to the agent within the scheduling process. The action space consists of several discrete types of decisions:

- $\mathcal{A}_0$: **Accept**. The agent accepts and begins processing a job located at position γ_r on the conveyor or at the workbench. If the job is on the conveyor, it is picked up and moved to the workbench; otherwise, if the job is already at the workbench, the next processing step is immediately executed.
- $\mathcal{A}_1$ to $\mathcal{A}_{w-1}$: **Wait**. The agent decides to wait for jobs arriving in sections $\gamma_r - 1$ through $\gamma_r - w + 1$. This action allows the agent to potentially gain a higher reward if waiting is more beneficial than processing the current job.
- $\mathcal{A}_d$: **Decline**. The agent rejects the job located directly in front of it and does not wait for any incoming jobs within its observation window.
- $\mathcal{A}_c$: **Continue**. The agent proceeds without interruption when no job is available in its window section or when it is currently processing a job.

3. **Reward (R)**: Based on the optimization mathematical model presented in subsection 2.2, the reward function for agent m at time t is formulated as:

$$R_t^m = w_a R_{\text{done}}(t) + w_b R_{\text{process}}^m(t) + w_c R_{\text{wait}}^m(t), \tag{11}$$

where the main objective is to minimize energy consumption, and w_a, w_b, w_c are positive scaling constants. It is recommended to configure these constants to satisfy the condition $w_a > w_b > w_c$. We asssign $w_a = 5.0$, $w_b = 2.0$, and $w_c = 0.5$ for training the MARL algorithm. The components of the reward function are defined as follows:

- $R_{\text{done}}(t)$ is the shared reward assigned to all agents based on the number of operations completed at time t:

$$R_{\text{done}}(t) = \hat{O}_{\text{done},t}, \tag{12}$$

where $\hat{O}_{\text{done},t}$ denotes the number of completed operations at timestep t.

- $R^m_{\text{process}}(t)$ is the individual reward given if agent m is processing a job. This component directly implements the objective function Equation (1) as a cost:

$$R^m_{\text{process}}(t) = \begin{cases} \dfrac{v^m_o}{\sum_{i=1}^{|M^o|} v^i_o} - \eta_{\text{q}}, & \text{if } \mathcal{G}^m_t = 2 \\ -\eta_{\text{p}}, & \text{otherwise,} \end{cases} \tag{13}$$

  At every timestep, the agent receives a negative reward (penalty) of η_q if it is working ($\mathcal{G}^m_t = 2$) or η_p if $\mathcal{G}^m_t \neq 2$. Additionally, the processing speed ratio of agent m (v^m_o) relative to the total processing speed of all agents M_o capable of performing operation type o. This formula encourages agents to work efficiently and pick their tasks wisely. It gives a higher reward to faster agents, but the penalties for both working and staying idle force the agents to decide if a specific task is actually worth the effort in the long run.
- $R^m_{\text{wait}}(t)$ is the reward given when agent m selects the Wait action:

$$R^m_{\text{wait}}(t) = \begin{cases} \dfrac{v_{m,o}}{x \sum_{i=1}^{|M^o|} v_{i,o}}, & \text{if action Wait is chosen} \\ 0, & \text{otherwise,} \end{cases} \tag{14}$$

  where x indicates the number of conveyor segments between the agent's position and the target job. This formulation accounts travel time and encourages workers to wait for closer jobs that are better for them in the long run.

3. Method

This section describes the proposed Multi-Agent Shared DQN framework designed for energy-efficient scheduling. We detail the implementation of the CTDE paradigm and the integration of action masking to ensure agents adhere to physical and logical workshop constraints. This architecture aims to enhance scalability and coordination while mitigating the challenges of partial observability.

3.1 Multi-Agent DQN for FJSSP

One of the DRL algorithms is the Deep Q Network (DQN), which is designed to estimate Q values for various states and actions. In a standard DQN architecture [23, 24], the input layer receives a state representing the current environmental condition, while hidden layers perform nonlinear transformations using weights and activation functions to maintain the network parameters. The output layer then calculates the estimated Q values for every possible action available within a specific state. As detailed in Algorithm (1), this research applies a Multi-Agent model based on the DQN framework that utilizes a single main network and a single target network shared among all agents to optimize scheduling performance. All experiences from each agent are collected into a single replay buffer to update the network parameters, while action selection is performed independently by each agent. During the training process,

Algorithm 1 Multi-Agent Shared Q-Network

INITIALIZE: replay buffer D; main network Q with parameters θ; target network Q^- with parameters $\theta^- \leftarrow \theta$.

for each episode = 1 to E **do**
  Reset the environment.
  while not done **do**
    for each agent m **do**
      Observe s_t^m and masking F_t^m from Algorithm (2)
      Select action a_t^m:

$$a_t^m \leftarrow \begin{cases} \text{random action, with probability } \epsilon \\ \pi^*(s,a), \text{ with probability } (1-\epsilon). \end{cases}$$

      Observe s_{t+1}^m, F_{t+1}^m, and reward R_t^m.
    end for
    Compute average reward: $R_t \leftarrow \frac{1}{M}\sum_{m=1}^{M} R_t^m$.
    Store $(s_t^m, F_t^m, a_t^m, R_t, s_{t+1}^m, F_{t+1}^m)$ into D for each agent m.

    TRAINING ALL AGENTS SIMULTANEOUSLY:
    for each agent m **do**
      Sample N transitions $(s_d^m, F_d^m, a_d^m, R_d, s_{d+1}^m, F_{d+1}^m)$ from D.
      Compute target:

$$y_d^m \leftarrow \begin{cases} R_d, & \text{if } s_{d+1}^m \text{ is terminal} \\ Q_{target}, & \text{otherwise} \end{cases}$$

    end for
    Compute loss:
      $L(\theta) \leftarrow \frac{1}{M}\sum_{m=1}^{M}\left(L^M(\theta)\right)$
    Update θ by gradient descent on $L(\theta)$.
    Update target network: $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$.
  end while
end for

each agent observes the environment, selects actions using an ϵ-greedy strategy, and receives rewards that are subsequently averaged. The transitions obtained are stored in the replay buffer and used to calculate targets based on the Bellman equation. The loss function is calculated as the average mean squared error of all agents, and network parameters are updated via the gradient descent algorithm, followed by a soft update on the target network. This model is designed to accelerate computation time by using only one main network, one target network, and one shared replay buffer. Nevertheless, each agent can still access status information from other agents available within each state.

$$\pi^*(s,a) = \arg\max_{a^m} Q^m(s_t^m, F_t^m, a^m; \theta^m), \tag{15}$$

$$Q_{target} = R_d + \gamma \max_{a^m} Q^-(s_{d+1}^m, F_{d+1}^m, a^m; \theta^-) \tag{16}$$

$$L^M(\theta) = \frac{1}{N}\sum_{d=1}^{N}\left(y_d^m - Q(s_d^m, F_d^m, a_d^m; \theta)\right)^2 \tag{17}$$

Equation (15) represents the decentralized execution policy $\pi^*(s, a)$, where each agent m independently selects an action a^m by maximizing its local action-value function Q^m given its state s_t^m and masking vector F_t^m. Equation (16)computes the temporal difference target Q_{target} using a target network Q^- with parameters θ^- to enhance training stability, following the standard Bellman optimality equation. Equation (17) defines the centralized loss function $L^M(\theta)$ as the mean squared error (MSE) between the target values y_d^m and the predicted Q-values across a mini-batch of N transitions, facilitating simultaneous parameter updates for all agents via gradient descent.

3.2 Action Masking

In the Algorithm (2), an action masking mechanism is implemented to prevent agents from selecting unsuitable actions based on their local status and individual capabilities. Masking functions by filtering out non-executable actions, such as accepting a job with an operation type that does not match the agent's capabilities or when no jobs are available in the agent's observation. This filtering process is conducted by setting the Q-value for invalid actions to $-\infty$, ensuring that these actions are not selected during the arg max process. Consequently, only strictly valid actions are considered during both the exploitation and exploration phases.

Algorithm 2 Dispatching Rules for Action Masking

```
for each agent m do
    INITIALIZE: accept ← 0, wait ← 0, decline ← 0, continue ← 0;
    Get states of all agents;
    if there is a job in the agent's observation window and job operation type matches agent capabilities
    then
        decline ← 1;
        wait ← 1;
        if job is on the conveyor next to the agent then
            accept ← 1;
        end if
    else
        continue ← 1;
    end if
    F_t^m ← [accept, wait, decline, continue];
end for
return F_t^m for each agent;
```

4. Experiments and Discussion

This section evaluates the performance and robustness of the MARL model through simulated production scenarios. We conduct a comparative reward analysis of various DQN architectures and assess the proposed model's efficiency in terms of energy consumption and makespan. Furthermore, the reliability of the system is measured by calculating success rates under stringent operational and energy-efficiency constraints.

Table 3. Environment Setup and Training Parameter

Parameter	Description
n_m	3
ω	2
n_ϵ	12
$\mathcal{N}_J$	21
Set of capable machines for operation type o	$M_o^1 = 1, 2;$ $M_o^2 = 2, 3;$ $M_o^3 = 1, 3$
speed of machines (ν^m)	$\nu^1 = 1;$ $\nu^2 = 2;$ $\nu^3 = 3$
Neuron hidden layer	256×128
Activation function hidden layer	Relu
Activation function output layer	linear
Batch size	128
Epsilon initial (ϵ)	1
Epsilon decay	0.992
Buffer size	1×10^5

4.1 Evaluation Indicators and Initial Initialization

This subsection discusses the indicators used to evaluate the performance of the test algorithms, the initialization of parameters within the experimental environment, and the specific algorithms applied in the analysis. The metrics utilized to evaluate the performance of the flexible job shop system are as follows:

1. **Energy Consumption in *units***: Energy consumption refers to the total energy utilized by each robot during the scheduling process, which includes conditions where the robot is active but idle as well as when it is actively performing tasks. Energy values are calculated cumulatively at each time step and expressed in units. This metric is the primary objective function defined in Equation 1 which evaluates system operational efficiency through the minimization of total energy consumption.
2. **Makespan in *steps***: Makespan is defined as the total duration required to complete all jobs from the initial start until the final completion of the scheduling process. This metric is measured in time steps which represent discrete time units within the simulation environment. Although the primary objective of this study is the minimization of energy, makespan functions as a vital performance indicator to observe the time efficiency achieved while prioritizing energy conservation.

To evaluate the performance of the MARL algorithm, this research excludes comparisons with alternative frameworks based on reinforcement learning. Instead, the proposed system is compared solely against heuristic and metaheuristic methodologies

to determine job acceptance and processing decisions. The selected benchmarks include First Come First Processed (FCFS) and Fastest Available Agent (FAA) as heuristic methods, alongside the Genetic Algorithm (GA) as a metaheuristic method. For the GA configuration, the population size is set to 30 and the number of generations is 50. The training parameters and environment configurations for this study are detailed in Table 3.

4.2 Comparative Reward Analysis of Multi-Agent Shared and Independent DQN

This subsection aims to conduct a comparative analysis of the learning performance of three Multi-Agent Deep Q-Network (DQN) architectures: Independent DQN, Shared DQN without shared status, and Shared DQN with shared status. In the Shared DQN with shared status model, agents not only share the Q-network parameters but also exchange status information, thereby facilitating enhanced coordination in decision-making. Conversely, in the Shared DQN without shared status model, agents share network parameters without any exchange of status information. Finally, in the Independent DQN model, each agent maintains a distinct Q-network and operates without any information exchange with other agents. The focus of this analysis is to evaluate the effectiveness of each model in maximizing cumulative reward during the training phase. Furthermore, this analysis examines performance stability and the impact of the shared status mechanism on inter-agent coordination. Through these experiments, the study aims to identify the optimal Multi-Agent model for a FJSSP.

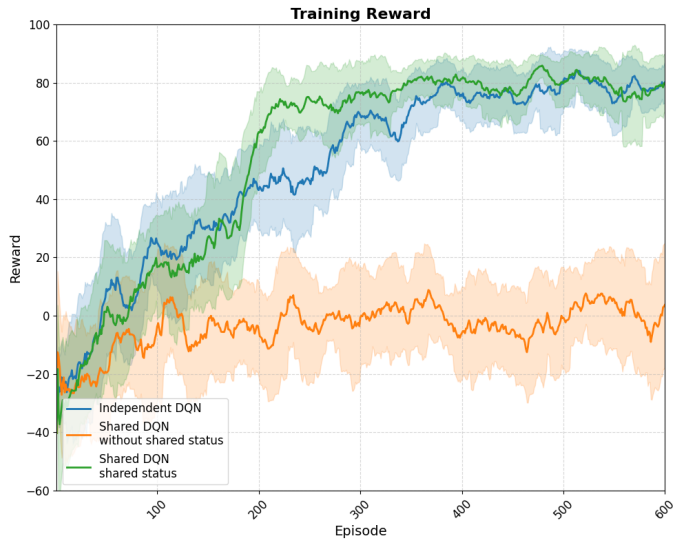

Figure 2. Reward Training

Figure 2 illustrates the training performance comparison across all Multi-Agent Deep Q-Network variants. The Shared DQN with shared status demonstrates superior performance, characterized by the most rapid and stable reward convergence. By sharing both Q-network parameters and inter-agent status information, this method

facilitates enhanced coordination. This is evidenced by a significant reward increase starting at episode 200, converging between 80 and 90 from episode 300 onwards. In contrast, the Shared DQN without shared status exhibits the poorest performance, maintaining low reward values with no significant improvement throughout the 600 training episodes. Meanwhile, the Independent DQN, where agents operate without information sharing, attains relatively high rewards after episode 300. However, it is less efficient than the status sharing model regarding computational overhead and memory usage. Specifically, Independent DQN necessitates separate gradient descent updates for each agent's distinct Q-network. Furthermore, it lacks memory efficiency as each agent requires an individual replay buffer to store interaction experiences.

These training results demonstrate that explicit collaboration, facilitated by sharing Q network parameters and interagent status information, not only enhances computational efficiency but also yields more stable and reliable policies within Multi-Agent systems. Consequently, the Shared DQN with shared status architecture is selected as the primary model for the subsequent Multi-Agent Reinforcement Learning (MARL) experiments detailed in the following subsection.

4.3 Algorithm Performance Analysis Based on Makespan and Energy Consumption

This subsection aims to evaluate the performance of the MARL algorithm regarding makespan and energy consumption. Within the flexible job shop system, the makespan denotes the total time in steps required to complete all tasks from start to finish within a single production cycle. The total evaluation involves testing across 1000 samples.

Table 4. Energy Consumption in *units* and Percentage of Energy Efficiency Improvement against Heuristic and metaheuristic Methods

Method	Outlier (%)	Min	Max	Average	Against FCFS	Against FAA
MARL	3.3%	244	266	250 ± 3	40.62%	35.23%
FCFS	0.2%	351	522	421 ± 26	–	–
FAA	2.1%	327	464	386 ± 23	–	–
GA	0.3%	335	493	400 ± 24	4.75%	-3.78%

Table 5. Makespan in *steps* and Percentage of Time Efficiency Improvement against against Heuristic and metaheuristic Methods

Method	Outlier (%)	Min	Max	Average	Against FCFS	Against FAA
MARL	1.0%	213	274	237 ± 11	1.66%	-0.42%
FCFS	0.4%	197	295	241 ± 16	–	–
FAA	1.3%	191	302	236 ± 17	–	–
GA	0.6%	197	295	241 ± 16	-0.08%	-2.20%

According to Table 4, the Multi-Agent Shared DQN method is significantly more efficient in energy consumption compared to the heuristic approaches (FCFS

and FAA) and the metaheuristic GA. The average energy consumption for MARL is 250 ± 3 units, which is substantially lower than the values for FCFS at 421 ± 26, FAA at 386 ± 23, and GA 400 ± 24 units respectively. While GA shows a slight improvement of 4.75% over FCFS, it consumes more energy than FAA by 3.78%. Consequently, GA energy-optimization results appear very similar to those of heuristic methods. MARL remains the superior choice for energy efficiency, achieving improvements of 40.62% relative to FCFS and 35.23% relative to FAA. Furthermore, the extremely narrow range between the minimum and maximum energy values for MARL signifies high operational reliability compared to the much wider standard deviations observed in GA and the heuristic methods.

According to Table 5, the MARL method demonstrates competitive makespan performance against all benchmarks. MARL achieves an average improvement of 1.66% over FCFS and GA, both of which share an identical average makespan of 241 ± 16 steps. Although FAA achieves the lowest average makespan at 236 steps, it exhibits higher variability as evidenced by a broader range of values and the highest standard deviation of ± 17. In contrast, GA performs similarly to FCFS and fails to outperform MARL or FAA in this specific environment. MARL maintains greater stability than both the metaheuristic and heuristic approaches, with a maximum makespan of only 274 steps.

Overall, MARL offers a superior balanced solution between task completion duration and energy efficiency. While FAA might achieve a faster completion time in specific instances and GA provides a slight energy advantage over FCFS, MARL provides a much more predictable and energy-efficient schedule. This consistency across multiple metrics establishes the superiority of the MARL approach over traditional heuristic and metaheuristic methods in a flexible job shop environment.

4.4 Algorithm Performance Analysis Based on Success Rates Under Stringent Constraints

This subsection evaluates the reliability of the MARL algorithm by measuring success rates under increasingly stringent operational constraints. The specific constraints are defined as follows:

1. The maximum conveyor capacity (n_{cap}) in Eq.(2) is set to 60% or 80%.
2. The maximum production time limit ($T_{\max}$) in Eq.(4) is set to 250 step.
3. The power consumption ($\mathcal{P}$) in Eq.(5) is set to 1.2 unit/step.
4. The total target number of jobs to be completed ($\mathcal{N}_J$) in Eq.(7) is set to 21 products.

For constraint in Eq. (3) and Eq. (6) represent hard constraints that are consistently satisfied during every production cycle. Within this framework, a successful production cycle is defined as the completion of all 21 products in less than 250 steps. The analysis investigates the capability of the proposed model to maintain high performance while adhering to conveyor capacity limits and energy consumption thresholds across 1000 randomized test samples. By comparing these outcomes against heuristic and metaheuristic methods, this section demonstrates the robustness of the MARL approach in complex manufacturing environments.

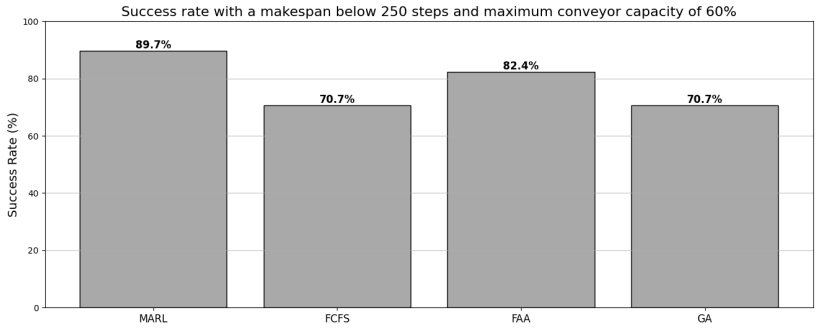

Figure 3. Percentage Success Rate with a Makespan below 250 Steps and 60 Percent Maximum Conveyor Capacity

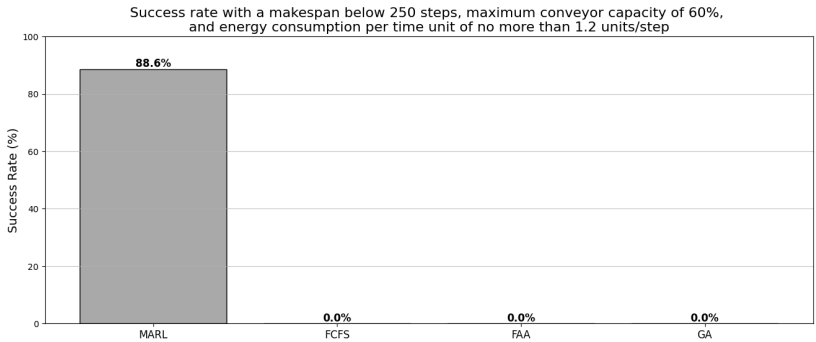

Figure 4. Percentage Success Rate with a Makespan below 250 Steps, 60 Percent Maximum Conveyor Capacity, and a Maximum Power Consumption of 1.2 Units per Step

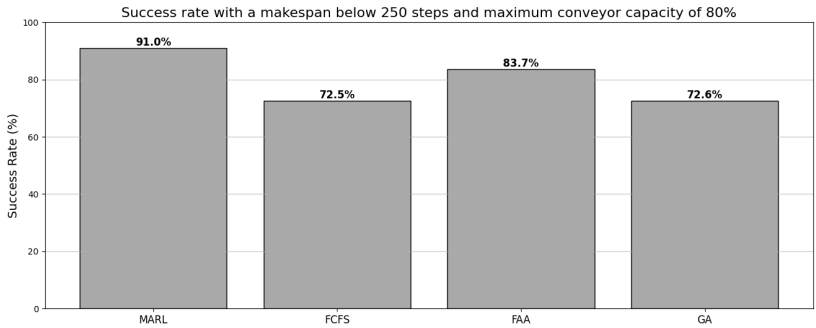

Figure 5. Percentage Success Rate with a Makespan below 250 Steps and 80 Percent Maximum Conveyor Capacity

Figure 3 illustrates the success rate comparison for n_{cap} = 60 percent. MARL demonstrates superior performance with a success rate of 89.7 percent, surpassing the 82.4 percent achieved by FAA and the 70.7 percent recorded by both FCFS

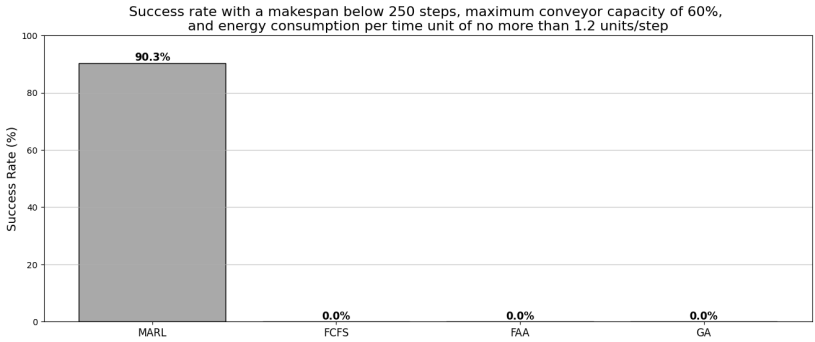

Figure 6. Percentage Success Rate with a Makespan below 250 Steps, 80 Percent Maximum Conveyor Capacity, and a Maximum Power Consumption of 1.2 Units per Step

and GA. These results confirm that MARL effectively manages dynamic job shop scheduling by meeting production targets while maintaining high efficiency. Notably, the metaheuristic GA does not show any performance advantage over the simpler FCFS heuristic in this specific scenario.

Figure 4 presents the success rate with an added constraint $\mathcal{P}$ = 1.2 units per step. This threshold evaluates the operational energy efficiency of the system in completing tasks punctually. Under these stricter constraints, MARL maintains a high success rate of 88.6 percent. Heuristic and metaheuristic methods fail entirely to satisfy these joint constraints, yielding a success rate of 0.0 percent. This outcome indicates that these approaches lack the flexibility to optimize time and energy simultaneously. Without an adaptive learning mechanism, these methods produce energy intensive scheduling patterns that cannot adhere to strict efficiency requirements.

Figure 5 and Figure 6 show a similar trend with n_{cap} = 80 percent. This higher threshold creates a higher density manufacturing environment where the conveyor accommodates a greater volume of concurrent tasks. As illustrated in Figure 5, MARL achieves a success rate of 91.0 percent at this higher capacity level. This represents a slight improvement over the 60 percent capacity scenario, suggesting the MARL agent effectively utilizes the additional conveyor space to enhance scheduling. While FCFS, FAA, and GA show marginal improvements to 72.5 percent, 83.7 percent, and 72.6 percent respectively, they remain unable to match the optimization capabilities of MARL approach. The robustness of the MARL model is further evidenced in Figure 6 with the power consumption constraint. MARL maintains a success rate of 90.3 percent, while all other benchmark methods, including the GA metaheuristic, fail completely with a success rate of 0.0 percent. The performance of the GA metaheuristic is very similar to FCFS across all scenarios, indicating that metaheuristic optimization is not suitable for the FJSSP within this specific scenario.

Overall, these results confirm that the MARL model successfully optimizes production time while minimizing energy consumption per unit of time. This adaptability establishes MARL as a superior and more reliable approach for modern manufacturing systems compared to heuristic and metaheuristic scheduling methodologies.

5. Conclusion

This study demonstrates that the MARL framework utilizing a Shared DQN effectively addresses the complexities of the flexible job shop scheduling problem by optimizing the trade off between production time and energy consumption. Experimental results indicate that the proposed architecture achieves significant energy efficiency improvements of 40.62 percent and 35.23 percent compared to FCFS and FAA respectively. The MARL approach also demonstrates superior efficiency over the metaheuristic, which exhibits energy consumption patterns similar to the heuristic methods. Furthermore, the model maintains exceptional operational stability with a standard deviation that is significantly lower than heuristic and metaheuristic methods across 1000 samples. Regarding time efficiency, MARL achieves a competitive makespan that is similar to the results produced by rule based approaches while ensuring greater predictability in completion times. The model also exhibits remarkable robustness under stringent constraints, whereas heuristic and metaheuristic methods yield zero percent success. These improvements are attributed to explicit coordination through shared network parameters and agent status information which facilitate more informed decision making in high density manufacturing environments. Consequently, the MARL approach establishes itself as a highly adaptive and sustainable solution for modern industrial scheduling.

References

[1] Fangfang Zhang et al. "Survey on Genetic Programming and Machine Learning Techniques for Heuristic Design in Job Shop Scheduling". In: *IEEE Transactions on Evolutionary Computation* 28.1 (2024), pp. 147–167. DOI: 10.1109/TEVC.2023.3255246.

[2] Yaping Fu et al. "Distributed scheduling problems in intelligent manufacturing systems". In: *Tsinghua Science and Technology* 26.5 (2021), pp. 625–645. DOI: 10.26599/TST.2021.9010009.

[3] Ling Wang, Zixiao Pan, and Jingjing Wang. "A Review of Reinforcement Learning Based Intelligent Optimization for Manufacturing Scheduling". In: *Complex System Modeling and Simulation* 1.4 (2021), pp. 257–270. DOI: 10.23919/CSMS.2021.0027.

[4] Zihui Luo et al. "Flow-Shop Scheduling Problem With Batch Processing Machines via Deep Reinforcement Learning for Industrial Internet of Things". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 8.5 (2024), pp. 3518–3533. DOI: 10.1109/TETCI.2024.3402685.

[5] Ruiqi Chen, Wenxin Li, and Hongbing Yang. "A Deep Reinforcement Learning Framework Based on an Attention Mechanism and Disjunctive Graph Embedding for the Job-Shop Scheduling Problem". In: *IEEE Transactions on Industrial Informatics* 19.2 (2023), pp. 1322–1331. DOI: 10.1109/TII.2022.3167380.

[6] Chien-Liang Liu, Chuan-Chin Chang, and Chun-Jan Tseng. "Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems". In: *IEEE Access* 8 (2020), pp. 71752–71762. DOI: 10.1109/ACCESS.2020.2987820.

[7] Chien-Liang Liu, Chun-Jan Tseng, and Po-Hao Weng. "Dynamic Job-Shop Scheduling via Graph Attention Networks and Deep Reinforcement Learning". In: *IEEE Transactions on Industrial Informatics* 20.6 (2024), pp. 8662–8672. DOI: 10.1109/TII.2024.3371489.

[8] Kun Lei et al. "Large-Scale Dynamic Scheduling for Flexible Job-Shop With Random Arrivals of New Jobs by Hierarchical Reinforcement Learning". In: *IEEE Transactions on Industrial Informatics* 20.1 (2024), pp. 1007–1018. DOI: 10.1109/TII.2023.3272661.

[9] Yejian Zhao et al. "Dynamic Jobshop Scheduling Algorithm Based on Deep Q Network". In: *IEEE Access* 9 (2021), pp. 122995–123011. DOI: 10.1109/ACCESS.2021.3110242.

[10] Kuo-Hao Ho et al. "Residual Scheduling: A New Reinforcement Learning Approach to Solving Job Shop Scheduling Problem". In: *IEEE Access* 12 (2024), pp. 14703–14718. DOI: 10.1109/ACCESS.2024.3357969.

[11] Stéphane Dauzère-Pérès et al. "The flexible job shop scheduling problem: A review". In: *European Journal of Operational Research* 314.2 (2024), pp. 409–432. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2023.05.017.

[12] Kun Lei et al. "A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem". In: *Expert Systems with Applications* 205 (2022), p. 117796. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.117796.

[13] Anbang Liu et al. "Integrating Machine Learning and Mathematical Optimization for Job Shop Scheduling". In: *IEEE Transactions on Automation Science and Engineering* 21.3 (2024), pp. 4829–4850. DOI: 10.1109/TASE.2023.3303175.

[14] ChengRan Lin, ZhengCai Cao, and MengChu Zhou. "Learning-Based Grey Wolf Optimizer for Stochastic Flexible Job Shop Scheduling". In: *IEEE Transactions on Automation Science and Engineering* 19.4 (2022), pp. 3659–3671. DOI: 10.1109/TASE.2021.3129439.

[15] Wen Song et al. "Flexible Job-Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning". In: *IEEE Transactions on Industrial Informatics* 19.2 (2023), pp. 1600–1610. DOI: 10.1109/TII.2022.3189725.

[16] Junwen Ding et al. "A Novel Evolutionary Algorithm for Energy-Efficient Scheduling in Flexible Job Shops". In: *IEEE Transactions on Evolutionary Computation* 27.5 (2023), pp. 1470–1484. DOI: 10.1109/TEVC.2022.3222791.

[17] Liji Shen, Stéphane Dauzère-Pérès, and Söhnke Maecker. "Energy cost efficient scheduling in flexible job-shop manufacturing systems". In: *European Journal of Operational Research* 310.3 (2023), pp. 992–1016. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2023.03.041.

[18] Jianhua Wang, Chuanyu Wu, and Yongtao Peng. "Multi-objective scheduling for an energy-efficient flexible job shop problem with peak power constraint". In: *Applied Soft Computing* 167 (2024), p. 112330. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2024.112330.

[19] Moisés Santana Pereira et al. "Energy Estimation and Production Scheduling in Job Shop Using Machine Learning". In: *IEEE Access* 12 (2024), pp. 104177–104189. DOI: 10.1109/ACCESS.2024.3430218.

[20] Dazzle Johnson, Gang Chen, and Yuqian Lu. "Multi-Agent Reinforcement Learning for Real-Time Dynamic Production Scheduling in a Robot Assembly Cell". In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 7684–7691. DOI: 10.1109/LRA.2022.3184795.

[21] Aaron Hao Tan et al. "Deep Reinforcement Learning for Decentralized Multi-Robot Exploration With Macro Actions". In: *IEEE Robotics and Automation Letters* 8.1 (2023), pp. 272–279. DOI: 10.1109/LRA.2022.3224667.

[22] Yuxin Li et al. "Real-Time Scheduling for Flexible Job Shop With AGVs Using Multiagent Reinforcement Learning and Efficient Action Decoding". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 55.3 (2025), pp. 2120–2132. DOI: 10.1109/TSMC.2024.3520381.

[23] Kok-Lim Alvin Yau et al. "The Augmented Intelligence Perspective on Human-in-the-Loop Reinforcement Learning: Review, Concept Designs, and Future Directions". In: *IEEE Transactions on Human-Machine Systems* 54.6 (2024), pp. 762–777. DOI: 10.1109/THMS.2024.3467370.

[24] Xu Wang et al. "Deep Reinforcement Learning: A Survey". In: *IEEE Transactions on Neural Networks and Learning Systems* 35.4 (2024), pp. 5064–5078. DOI: 10.1109/TNNLS.2022.3207346.