Article

A Deep Reinforcement Learning Approach for FJSP Based on Job-Enhanced Heterogeneous Graph Neural Network

Jian Zhang[1,*], Yusen Zhao[1] and Cunrui Ying[2]

[1] School of Computer and Software, Southwest Petroleum University, Chengdu, China
[2] School of Civil Engineering and Geomatics, Southwest Petroleum University, Chengdu, China
* Correspondence: Jian Zhang, School of Computer and Software, Southwest Petroleum University, Chengdu, China

Abstract: The Flexible Job Shop Scheduling Problem (FJSP) is a critical combinatorial optimization problem in modern manufacturing systems, aiming to simultaneously determine optimal machine assignments and operation sequences. Traditional deep reinforcement learning (DRL) methods often face significant challenges in effectively modeling complex global constraints and designing adaptive action selection mechanisms for highly dynamic environments. To address these critical limitations, this paper proposes a novel end-to-end deep reinforcement learning framework based on the Proximal Policy Optimization (PPO) algorithm. By strategically introducing job nodes into the traditional operation-machine heterogeneous graph, the proposed framework constructs an innovative three-type-node heterogeneous graph consisting of jobs, operations, and machines, thereby significantly enhancing its representation ability for intricate scheduling constraints. Furthermore, to address the dynamic nature of the action space inherent in FJSP, the framework adaptively handles variable action dimensions by dynamically generating an interaction matrix via a multi-layer perceptron. It then calculates the priorities of feasible actions using a bilinear inner product, ultimately achieving highly adaptive and efficient decision-making. Extensive experimental results conducted on standard benchmark datasets demonstrate that the proposed framework achieves superior scheduling performance, including substantially reduced makespan and improved computational efficiency, compared to existing state-of-the-art methods. These findings comprehensively validate its effectiveness and robustness in solving complex, large-scale FJSP instances in real-world manufacturing scenarios.

Keywords: flexible job shop scheduling; deep reinforcement learning; graph neural network; job node embedding; bilinear interaction; adaptive decision-making

Received: 16 September 2026
Revised: 28 October 2026
Accepted: 12 November 2026
Published: 18 November 2026

Copyright: © 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The flexible job shop scheduling problem (FJSP) is a representative NP-hard combinatorial optimization problem, holding significant value in both academic research and industrial production [1]. Compared with the traditional job shop scheduling problem (JSP), FJSP permits each operation to be processed on multiple candidate machines and requires joint optimization of operation sequencing and machine assignment, resulting in substantially increased complexity and adaptability.

Current solution approaches for FJSP are broadly categorized into exact algorithms and approximate algorithms: exact algorithms guarantee optimal solutions but incur prohibitive computational cost, limiting their applicability to large-scale instances; approximate algorithms—including heuristic, meta-heuristic, and machine learning–based methods—are widely adopted due to their computational efficiency.

In FJSP, heuristic methods commonly employ composite scheduling rules that combine operation sequencing and machine assignment strategies—for example, first-in-first-out paired with shortest processing time. These methods offer rapid computation and straightforward implementation, enabling feasible solutions to be generated quickly.

However, their effectiveness heavily relies on rule design; no single rule consistently performs well across diverse and dynamic scheduling environments, leading to notable performance fluctuations across different problem instances. Meta-heuristic methods conduct global exploration of the solution space by emulating natural optimization principles and generally yield high-quality solutions. Representative examples include tabu search, particle swarm optimization, and genetic algorithms [2-4].

In recent years, deep reinforcement learning has gained increasing attention for addressing complex scheduling problems. One study trained a Dueling Double DQN–based policy model for adaptive job shop scheduling, representing the state as a three-channel matrix, using convolutional neural networks for feature extraction, and defining the reward function based on increments in machine utilization [5]. Another approach represented the state using a feasible solution matrix—capturing the sequence of policy selections without manual feature engineering—and incorporated machine idle time penalties into the reward function [6]. However, such matrix-based state representations are inherently tied to fixed problem dimensions and lack scalability across varying instance sizes. Consequently, graph neural networks have emerged as the dominant paradigm owing to their strong capability in modeling structural relationships.

One work proposed a multi-pointer graph network and introduced a hierarchical action space for FJSP to decouple job selection from machine selection [7]. Another addressed the distributed job shop scheduling problem by designing an action space grounded in operation-factory pairings to support coordinated decision-making, while constructing a spliced disjunctive graph representation and applying graph neural networks for state embedding [8]. A third study modeled the scheduling state as a heterogeneous graph, encoded operation node features via a heterogeneous graph neural network, unified job and machine decisions into a composite action, and employed the proximal policy optimization algorithm [9]. Yet another utilized a lightweight multi-layer perceptron as the state embedding network and designed a state matrix incorporating four key temporal features, achieving strong performance with reduced model complexity [10]. Building upon heterogeneous graph structure, a dual attention network composed of two distinct message-passing attention modules—one for operations and one for machines—was proposed to improve scheduling quality by capturing intricate inter-node dependencies [11]. An incremental scheduling strategy was also introduced, wherein scheduled nodes are removed after each decision step and heterogeneous graph neural networks are applied to refine representations of the remaining unscheduled elements [12].

Despite substantial advances in deep reinforcement learning–based methods for FJSP, two persistent challenges remain. First, the intricate many-to-many associations among operations and machines complicate accurate modeling of system-wide relational information using graph neural networks. Second, the requirement to jointly determine operation execution order and machine allocation leads to a feasible action set that evolves continuously with the scheduling state. To tackle these issues, this paper introduces an end-to-end deep reinforcement learning framework tailored for efficient FJSP resolution. The framework fully captures inter-node relational dependencies within heterogeneous graphs, supports arbitrary instance scales, and enables unified, context-aware selection among all dynamically admissible actions at each scheduling step. The main contributions of this work are as follows:

Job-enhanced heterogeneous graph representation: A heterogeneous graph comprising three node types—job, operation, and machine—is constructed by explicitly incorporating job-level nodes into the conventional heterogeneous graph structure, thereby strengthening the expressive power of the heterogeneous graph neural network.

Bilinear interaction–based action selection mechanism: An action selection architecture grounded in bilinear interaction is introduced. This architecture dynamically constructs an interaction matrix from global state features and applies it to compute the probability distribution over feasible actions, ensuring robust adaptation to the evolving action space in FJSP.

Comprehensive experimental validation: Systematic comparative experiments and ablation studies are conducted on 130 standard FJSP benchmark instances drawn from four widely used datasets, confirming the method's effectiveness, cross-instance generalization capability, and the functional necessity of each proposed component.

2. Problem Formulation and Graph Representation

Flexible Job Shop Scheduling Problem (FJSP): Given n jobs $J = \{J_1, J_2, \ldots, J_n\}$ and m machines $M = \{M_1, M_2, \ldots, M_m\}$, each job J_i consists of n_i operations with strict precedence requirements among these operations. Each operation O_{ij} can be processed on any machine in the optional machine set $M_{i,j} \subseteq M$, with the corresponding processing time denoted as P_{ijk}. Processing of any operation is non-preemptive once started, and each machine can execute only one operation at a time. The optimization objective is to determine both the machine assignment and the processing sequence for all operations to minimize the maximum makespan.

Disjunctive Graph Representation of the FJSP: The disjunctive graph is a standard graph model used to represent job shop scheduling problems, defined as $G = (O, C, D)$ [13]. The node set O includes all operation nodes along with a virtual source node and a virtual sink node. The conjunctive arc set C encodes the precedence requirements among operations within the same job, while the disjunctive arc set D captures the undetermined ordering relationships among operations assigned to the same machine. Unlike the classical Job Shop Scheduling Problem (JSP), a single operation in the FJSP may participate in multiple disjunctive arcs, and a feasible schedule corresponds to a directed acyclic graph formed by assigning directions to all disjunctive arcs. As illustrated in figure 1, figure 1a and 1b show the disjunctive graph structure and a feasible solution for a representative FJSP instance, respectively.

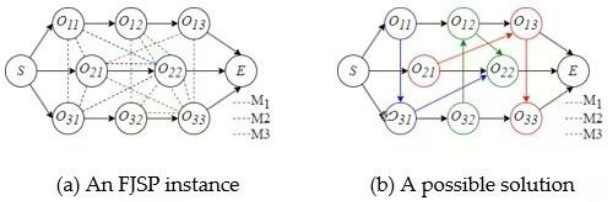


Figure 1. The Disjunctive Graph for FJSP.

3. Proposed Method

This section presents the proposed method. The flexible job shop scheduling problem (FJSP) is formulated as a sequential decision-making process, where the agent selects an operation and assigns it to a suitable machine based on the current state of the shop floor at each decision step. This procedure repeats iteratively until all operations are fully scheduled. The overall framework of the method is illustrated in figure 2.

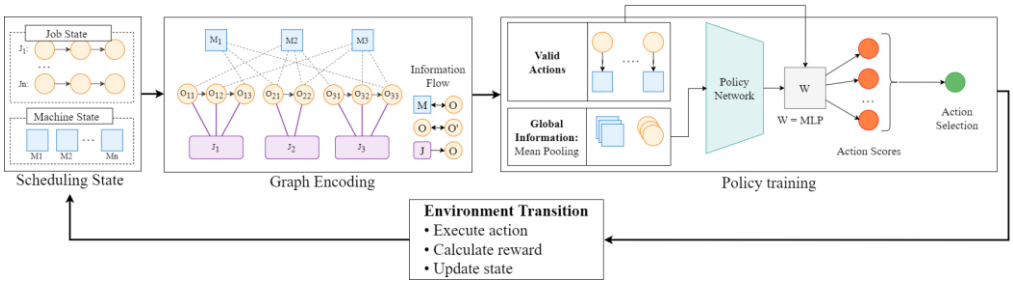


Figure 2. The Workflow of the Proposed Method.

At each decision step, the current scheduling state is first encoded as a heterogeneous graph structure (Section 3.2). A job-enhanced heterogeneous graph neural network

(HGNN) is then applied to this representation (Section 3.3): Job node embeddings are initialized once and remain fixed across layers, while operation and machine node embeddings are updated iteratively via a multi-layer message passing mechanism; and graph-level features are extracted using global pooling. Action selection probabilities for all feasible operation-machine pairs are computed through a bilinear interaction mechanism, and a scheduling action is sampled accordingly. Model training employs the Proximal Policy Optimization (PPO) algorithm (Section 3.4). Prior to these components, Section 3.1 introduces the Markov Decision Process (MDP) formulation underlying the entire process.

3.1. MDP Formulation

In reinforcement learning, the flexible job shop scheduling problem (FJSP) is typically modeled as a Markov Decision Process (MDP), defined by a 5-tuple(S, A, P, R, γ), where S denotes the state space, A the action space, P the state transition function, R the reward function, and γ the discount factor.

State. State st represents the scheduling state of the workshop at time t. Features of operation nodes include scheduling status, earliest start time, earliest completion time, position index within the job, number of eligible processing machines, waiting time, number of remaining operations in the job, and job completion progress. Features of machine nodes include the count of operations that can be processed, next available time, and utilization rate. Features of job nodes include completion progress, remaining processing time, total processing time, and completion status.

Action. The action is defined as $a_t = (O_{ij}, M_k)$, representing the assignment of operation O_{ij} to machine M_k for processing. Feasible actions must satisfy operational precedence requirements and machine capability requirements. As scheduling progresses, the set of feasible actions $A(s_t)$ evolves dynamically. To ensure selection exclusively from valid options, an action masking mechanism is applied during decision-making to exclude infeasible assignments.

Transition. Given current state st and action $a_t = (O_{ij}, M_k)$, the agent transitions to a new state s_{t+1} upon executing the action. This transition involves assigning operation O_{ij} to machine M_k, updating node attributes and graph structure, and recomputing the feasible action set. The scheduling process concludes when all operations have been fully processed.

Reward. Following standard practice in reinforcement learning–based scheduling approaches, the reward is defined as the negative increment in makespan. Specifically, if executing action at time t increases the maximum makespan from $C_{\max}(s_t)$ to $C_{\max}(s_{t+1})$, the step reward is given by formula (1). This design directly incentivizes reduction in makespan; thus, maximizing cumulative reward is equivalent to minimizing makespan, fully aligning the reinforcement learning objective with that of the FJSP.

$$r_t = -(C_{\max}(s_{t+1}) - C_{\max}(s_t)) = -\Delta C_{\max} \quad (1)$$

$$G = \sum_{t=0}^{T-1} r_t = -C_{\max}(s_T) \quad (2)$$

3.2. Heterogeneous Graph Structure

Due to the high scheduling flexibility of the flexible job shop problem, traditional disjunctive graphs are not well suited for representing and processing this problem. In this paper, a heterogeneous graph $G = (V, E)$ is adopted to represent the scheduling state, where the node set $V = V_O \cup V_M \cup V_J$ consists of operation, machine, and job nodes. The edge set E comprises three types: the operation-machine edge E_{OM} indicating feasible machine assignments for each operation, the operation-operation edge E_{OO} encoding the sequential order among operations, and the job-operation edge E_{JO} specifying the job affiliation of each operation. An example of the transformation from a disjunctive graph to a heterogeneous graph is illustrated in figure 3.

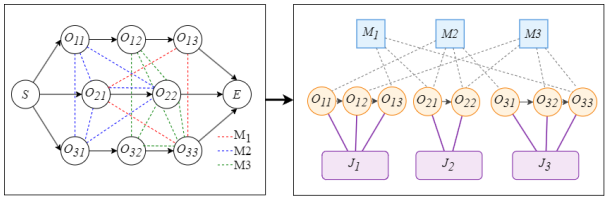


Figure 3. The Process of Transforming Disjunctive Graphs into Heterogeneous Graphs.

3.3. Job Enhanced HGNN

3.3.1. Job Enhanced Node Embedding

A job is a logical entity composed of multiple sequentially executed operations. Though not a physical entity, its state—such as completion progress and remaining workload—is essential for scheduling decisions. By introducing job nodes, operation nodes can indirectly access information about other operations within the same job through these intermediate nodes, thereby enabling broader information aggregation with fewer message-passing layers while mitigating over-smoothing effects.

Accordingly, the embedding of each job node integrates two components: aggregated operation features and intrinsic job attributes. For job Ji comprising n operations $O_{i1}, O_{i2}, O_{i3}, \ldots, O_{in}$, whose initial features are $h(O_{ij}) \in R^8$, the average of all operation features is computed and concatenated with the original job feature h(Ji), yielding the complete job embedding $h'(J_i) \in R^{12}$. During subsequent node updates, this job embedding propagates to operation nodes, allowing them to incorporate contextual information from other operations in the same job and thereby strengthening the global coherence of operation representations.

$$h'(J_i) = \text{Avg}(\{h(O_{i1}), h(O_{i2}), \ldots, h(O_{in})\}) \| h(J_i). \quad (3)$$

3.3.2. Node Update and Pooling

The heterogeneous graph is encoded using a heterogeneous graph neural network. Following job node embedding as described in Section 3.3.1, operation and machine node embeddings undergo L rounds of iterative updates. The update mechanisms for each node type are detailed below.

3.3.3. Updating of Machine Nodes

An attention mechanism is employed to update machine node embeddings. For machine M_k, the embedding is updated by aggregating information from neighboring operations and its own current state. Specifically, attention scores between the machine and each neighboring operation are first computed, enabling differentiated focus across operations and thereby yielding a more accurate characterization of machine load states.

$$e_{ijk} = \text{LeakyReLU} a^{\top} \left[W_M h(M_k) \| W_O h(O_{ij}) \right] \quad (4)$$

$$e_{kk} = \text{LeakyReLU} a^{\top} [W_M h(M_k) \| W_M h(M_k)] \quad (5)$$

$$h'(M_k) = \sigma(\alpha_{kk} W_M h(M_k) + \sum_{O_{ij} \in N(M_k)} \alpha_{ijk} W_O h(O_{ij})) \quad (6)$$

3.3.4. Updating of Operation Nodes

Operation nodes aggregate information from five sources: machine nodes, predecessor operations, successor operations, their own states, and job nodes, with a dedicated multilayer perceptron (MLP) designed for each source to perform feature transformation. For operation O_{ij}, its embedding update rule at the $l+1-th$ layer is shown in Formula (7): MLP_1 aggregates the sum of embeddings of all compatible machines M_k, MLP_2 and MLP_3 process the predecessor operation $O_{i(j-1)}$ and successor operation $O_{i(j+1)}$, respectively, MLP_4 encodes its own current-layer embedding $h^{(l)} O_{ij}$, and MLP_5 fuses the embedding $h^{(l)}(J_i)$ of its affiliated job node J_i; the five transformed features are concatenated and mapped through the Project layer to obtain the final embedding of the operation node.

$$h^{(l+1)}(O_{ij}) = \text{Project}[\text{MLP}_1(\sum_{M_k \in M(O_{ij})} h^{(l)}(M_k)) \| \text{MLP}_2 h^{(l)} (O_{i(j-1)}) \| \text{MLP}_3 h^{(l)}(O_{i(j+1)}) \| \text{MLP}_4 h^{(l)}(O_{ij}) \| \text{MLP}_5 h^{(l)}(J_i)] \quad (7)$$

where $M(O_{ij})$ denotes the set of compatible machines for O_{ij}. The hidden layer dimension of all $MLPs$ is set to 128, and the projection layer maps the concatenated features back to the standard embedding dimension.

3.3.5. Stacking and Pooling

To extract effective global features, the node update process is executed over Llayers. Job node embeddings are constructed only in the first layer and remain fixed thereafter; within each layer, updates proceed sequentially—first on machine nodes, then on operation nodes. Job nodes are excluded from global pooling, as job-level structural information has been fully incorporated into operation embeddings through the MLP modules. After L layers, the final machine embeddings $h_{M_k}^{(L)}$ and operation embeddings $h_{O_{ij}}^{(L)}$ are obtained.

After completing L layers of iteration and node updates, average pooling is applied separately to the machine embeddings and operation embeddings, and the pooled results are concatenated to obtain the global state representation:

$$h_g = \frac{1}{|O|}\sum_{O_{ij}\in O} h_{O_{ij}}^{(L)} \,\|\, \frac{1}{|M|}\sum_{M_k\in M} h_{M_k}^{(L)} \tag{8}$$

3.3.6. Bilinear Interactive Action Selection

For the current scheduling state s_t, let the candidate operation be O_{ij} and the compatible machine be M_k, where $A(s_t)$ denotes the set of feasible actions under state s_t. Figure 4 presents an illustrative example of action score calculation.



Figure 4. An Illustrative Example of Action Score Calculation.

$$W(s_t) = \mathrm{MLP}(h_g), \tag{9}$$

$$\mu(a_t|s_t) = h^{(L)}(O_{ij})^{\top} W(s_t) h^{(L)}(M_k) \tag{10}$$

$$\pi_\vartheta(a_t|s_t) = \frac{\exp(\mu(a_t|s_t))}{\sum_{a_t'\in A(s_t)} \exp(\mu(a_t'|s_t))}, \tag{11}$$

3.4. Training Algorithm

The Proximal Policy Optimization (PPO) algorithm is employed to train the proposed heterogeneous graph neural network (HGNN) scheduling model. The overall network architecture is illustrated in figure 5. Job nodes undergo feature construction solely in the initial embedding layer, whereas operation and machine nodes iteratively update their representations across L layers to capture high-order interaction patterns. Subsequently, operation and machine embedding features are aggregated into a global feature representation via average pooling. This global representation is then separately fed into the Critic and Actor networks. The Actor network maps the global features to a bilinear interaction matrix using a multilayer perceptron (MLP), enabling computation of matching scores for operation–machine pairs. Action probabilities are derived by applying softmax normalization to these scores.

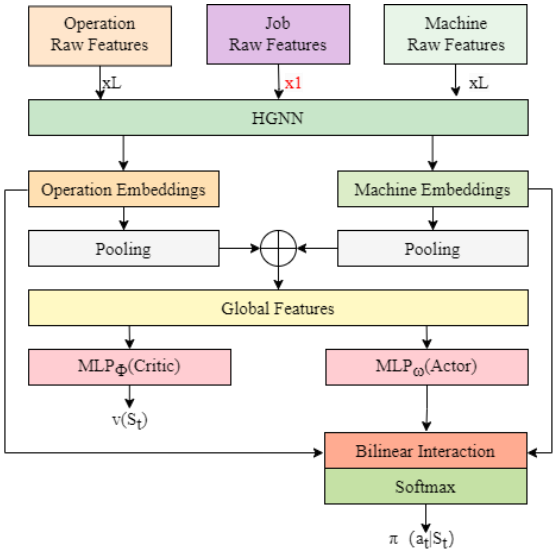


Figure 5. The Network Architecture.

Algorithm 1 Training Procedure with HGNN-PPO

Require: HGNN with parameter θ, Actor network with parameter ω, Critic network with parameter φ

Ensure: Trained policy π_ω

: **for** episode = 1, 2,...,E **do**
    for b = 1, 2,...,B {In parallel} **do**
        Initialize state s_0 based on FJSP instance b;
        while s_t is not terminal **do**
            Extract node embeddings using HGNN(θ);
            Compute action scores via bilinear interaction;
            Sample action $a_t \sim \pi_\omega(\cdot \mid s_t)$;
            Execute action and receive reward rt, next state s_{t+1};
            Store transition (s_t, a_t, r_t, s_{t+1});
            $s_t \leftarrow s_{t+1}$;
        end while
    end for
    Compute advantage estimates $\hat{A}t$ for collected trajectories;
    for k = 1, 2,...,K **do**
        Compute PPO loss $L(\theta, \omega, \varphi)$;
        Update parameters θ, ω and φ;
    end for
    if episode mod V = 0 **then**
        Evaluate policy on validation set;
    end if
    if episode mod R = 0 **then**
        Sample new batch of FJSP training instances;
    end if
: **end for**
: **return** Trained policy π_ω;

Training employs a parallel environment interaction mechanism, where B flexible job shop scheduling problem (FJSP) instances are processed concurrently to enable synchronous policy learning. The full training procedure is summarized in Algorithm 1. In each training iteration, the agent interacts with all parallel environments simultaneously and collects trajectory data for policy optimization. Network parameters are updated every T steps according to a fixed update interval. Policy performance is evaluated on a static validation dataset matching the training set in scale. During training,

actions are sampled stochastically from the current policy distribution $\pi_\theta(\cdot|s_t)$ to ensure adequate exploration; during testing, the action with the highest probability is selected deterministically for performance assessment.

4. Experiments

4.1. Experimental Setting

Network Architecture. The embedding dimension of operation nodes is set to 8; that of machine nodes is 8; and that of job nodes is 12. The number of iteration layers for the HGNN is set to L = 2, the hidden layer dimension of MLPs is 128, and the hidden layer dimension of the Actor network is 64.

Training Parameters. For the PPO algorithm, the learning rate is set to 0.0002 with the Adam optimizer adopted. The clip ratio is set to ε = 0.2, the discount factor γ = 1.0, and the entropy regularization coefficient is 0.01, and the value function loss coefficient is 0.5. Training is conducted with 20 parallel environments, with 3 PPO updates per round and a total of 1000 training rounds. The training instances are replaced every 20 rounds, and the model is validated and the optimal version is saved every 10 rounds.

Experimental Design. To ensure fairness and reproducibility, all comparative methods adopt a unified network architecture and training configuration. The training and validation datasets are identical to those used in the proposed method, and the random seed is fixed at 42. Upon acceptance, all implementation code and experimental data will be made publicly available.

Datasets: Training, Validation, and Test Sets. The training and validation sets consist of synthetic flexible job shop scheduling problem instances with 10 jobs and 5 machines, where each operation can be processed on an average of 4–5 machines. The training set contains 300 instances and the validation set 100 instances. For testing, standard benchmark instances are employed: 10 instances from the Brandimarte benchmark and 40 instances from each of the Hurink edata, rdata, and vdata subsets [14,15]. Flexibility increases across these subsets, and the test set spans problem scales from 10×6 to 30×10, comprising 130 instances to comprehensively assess performance and generalization capability.

Evaluation Metrics. Minimizing the makespan is the primary optimization objective, a key metric for evaluating scheduling effectiveness. Relative Percentage Deviation (RPD) is computed as follows, where C_{alg} denotes the makespan obtained by the algorithm, and C_{best} represents the currently known optimal or best-known solution. A smaller RPD value indicates better algorithm performance, and an RPD value of 0 signifies attainment of the optimal solution.

$$\text{RPD} = \frac{C_{\text{alg}} - C_{\text{best}}}{C_{\text{best}}} \times 100\%, \tag{12}$$

4.2. Experimental Results

To comprehensively verify the effectiveness of the proposed method, this paper compares it with various representative algorithms on public datasets, including two metaheuristic algorithms, four state-of-the-art deep reinforcement learning methods, and three classic heuristic rules.

Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) are adopted as baselines for metaheuristic algorithms, with a population size of 100 and 500 independent evolutions for each instance. For deep reinforcement learning methods, several recently published approaches are compared, and all methods are tested using a greedy policy after training. Three optimal combinations of heuristic rules—First-In-First-Out (FIFO) plus Earliest End Time (EET), Most Work Remaining (MWKR) plus EET, and Most Operations Remaining (MOPNR) plus EET—are selected.

A comprehensive performance comparison of all algorithms is conducted on various test sets, with detailed results presented in table 1.

Table 1. Comparison Results on FJSP Benchmark Datasets (Cmax and Gap Values)

Methods	Brandimarte		Hurink(edata)		Hurink(rdata)		Hurink(vdata)	
	Cmax	Gap(%)	Cmax	Gap(%)	Cmax	Gap(%)	Cmax	Gap(%)
GA	181.3	6.8	-	-	-	-	938.2	2.4
PSO	181.4	7.1	-	-	-	-	925.7	0.8
FIFO+EET	194.2	15.1	1250.5	21.2	1088.9	17.6	981.5	7.5
MWKR+EET	196.2	15.4	1483.6	44.0	1164.7	26.1	979.0	7.3
MOPNR+EET	203.0	21.6	1291.6	25.1	1111.3	19.9	995.3	9.1
Song et al. [9]	196.6	24.0	1196.0	16.2	1039.2	12.0	957.7	4.6
Wang et al. [11]	192.6	20.1	1180.0	15.1	1032.6	10.1	947.2	3.4
Yuan et al. [10]	186.8	13.2	1187.9	15.5	1041.8	12.1	963.5	5.4
Ho et al. [12]	213.6	39.2	1203.1	16.9	1065.3	14.8	1034.0	12.7
Ours	194.5	21.3	1175.3	14.4	1032.8	11.5	946.4	3.5

Metaheuristic algorithms (GA, PSO) attain the best solution quality owing to instance-wise independent optimization and global search in the solution space. Nevertheless, they suffer from heavy computational cost (2–3 minutes per instance) and are only evaluated on the Brandimarte and Hurink (vdata) datasets to establish the upper bound of solution performance. Heuristic rules achieve the highest computational efficiency but are significantly outperformed by intelligent optimization approaches in terms of solution quality.

Deep reinforcement learning methods exhibit stronger practicality, as policies learned from small-scale instances can generalize to unseen instances of arbitrary scales. Although slightly inferior to metaheuristics in solution quality, these approaches achieve inference speed comparable to heuristic rules, striking a favorable trade-off between optimality and efficiency, which makes them well suited for real-time scheduling scenarios.

Table 1 shows that the proposed method exhibits stable and excellent performance across different datasets: on the Brandimarte dataset, it achieves an average relative percentage deviation (RPD) of 21.3% (ranking third), with a marginal gap of only 1.2 percentage points from the second-ranked method (20.1%), while another method shows outstanding performance with an average RPD of 13.2%; on the Hurink (edata) dataset, the proposed method attains an average RPD of 14.4%, outperforming the second-best method (15.1%) by 0.7 percentage points and achieving the optimal performance; on the Hurink (rdata) dataset, it has an average makespan of 1032.8, differing by only 0.2 from the optimal method (1032.6), with nearly identical performance; on the Hurink (vdata) dataset, the proposed method obtains the lowest average makespan (946.4), yet its average RPD (3.5%) is slightly higher than that of the best-performing method (3.4%). This phenomenon arises from the computational characteristics of RPD: when the theoretical lower bound of the makespan is small, even a tiny absolute difference in makespan will result in a more pronounced impact on the relative percentage, which is particularly evident in small-scale instances.

The proposed method performs excellently on all four standard test sets, achieving optimal or near-optimal results on multiple datasets, and attains the lowest overall average makespan among all deep reinforcement learning methods. Its stable performance across datasets fully verifies its effectiveness and robustness.

The superior performance of the proposed method can be mainly attributed to two factors. First, the introduction of job-level global information into the heterogeneous graph neural network enhances the state representation capability, enabling the model to better preserve inter-node relationships and global scheduling states during graph propagation and pooling. Second, the bilinear interaction-based action selection mechanism allows the model to make more accurate decisions for instances with varying scales and complexities.

For a more intuitive visualization of the performance differences among various methods, the comparison results of all deep reinforcement learning approaches are presented in figure 6. It can be observed that the proposed method achieves stable and highly competitive performance across all test sets, reaching or approaching the optimal performance on multiple datasets, which further validates its robustness and effectiveness.

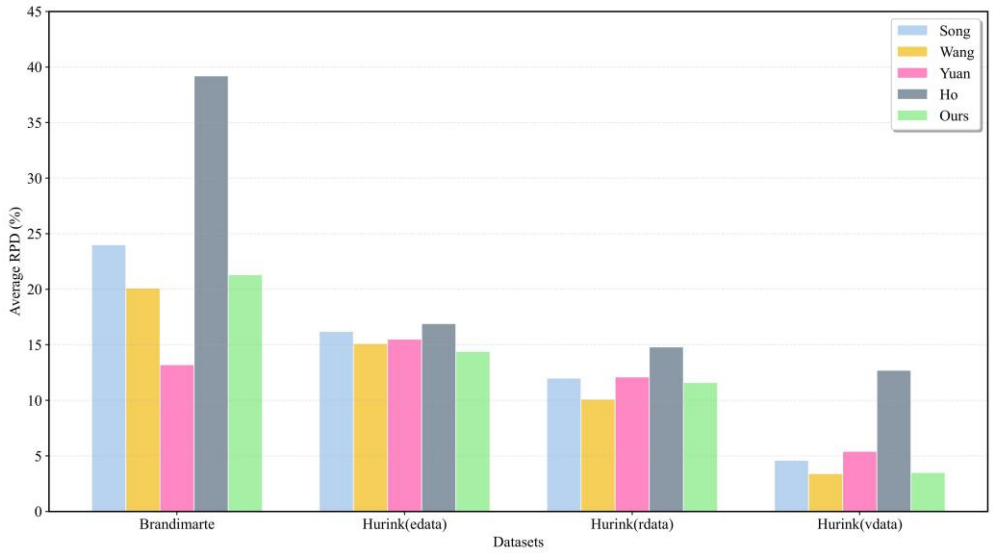


Figure 6. Comparison of Average RPDs with Other Learning-Based DRL Methods.

4.3. Ablation Experiment

To conduct an in-depth analysis of the effectiveness of the proposed core modules, this paper designs a systematic ablation study and compares four types of model variants: Raw denotes the baseline model without any newly added modules, Bilinear represents the variant with only the bilinear interaction action selection mechanism introduced, Job stands for the variant with only the job node embedding mechanism introduced, and All is the complete model integrating both modules simultaneously. The training processes of each model are illustrated in figure 7, and the experimental results on the standard test sets are presented in table 2.

Table 2. Ablation Experiment Results on FJSP Datasets.

Methods	Brandimarte		Hurink(edata)		Hurink(rdata)		Hurink(vdata)	
	Cmax	Gap(%)	Cmax	Gap(%)	Cmax	Gap(%)	Cmax	Gap(%)
Raw	200.2	25.6	1194.8	15.8	1037.8	11.2	966.6	5.3
Bilinear	198.0	24.6	1193.8	16.2	1040.4	12.2	959.3	4.8
Job	198.1	24.4	1192.0	15.7	1029.6	10.9	948.3	3.6
All	194.5	21.3	1175.3	14.4	1032.8	11.5	946.4	3.5

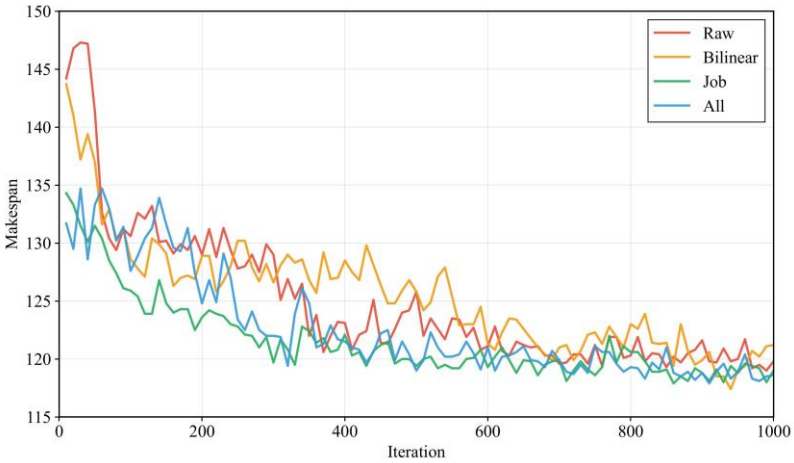


Figure 7. Comparison of Training Curves for Different Model

Although the makespan values of all models are close after convergence, the Job and All models exhibit faster convergence speeds and higher training stability, which verifies the positive contributions of the proposed modules to scheduling performance.

Compared with Raw, Job reduces both mean makespan and mean RPD on all four benchmark sets. Bilinear reduces both metrics on Brandimarte and Hurink vdata. On Hurink edata, Bilinear reduces mean makespan but increases mean RPD. On Hurink rdata, both metrics increase. Compared with Raw, All reduces mean makespan from 1194.8 to 1175.3 and mean RPD from 15.8% to 14.4% on Hurink edata. On Hurink rdata, All reduces mean makespan from 1037.8 to 1032.8, while mean RPD increases from 11.2% to 11.5%. Among the four variants, All achieves the lowest values for both metrics on Brandimarte, Hurink edata, and Hurink vdata, while Job achieves the lowest values on Hurink rdata.

5. Conclusion

This paper proposes an end-to-end deep reinforcement learning framework for the flexible job shop scheduling problem based on the proximal policy optimization algorithm. A three-type-node heterogeneous graph is constructed—comprising job, operation, and machine nodes—by introducing explicit job nodes into the traditional operation-machine graph. A job node embedding mechanism is designed to fuse operation-aggregated features with raw job attributes, enabling operation nodes to incorporate global information associated with their parent jobs. This enhances the graph's capacity to model scheduling requirements. To handle the dynamically varying action space, a global feature-driven bilinear interaction mechanism is introduced: a multilayer perceptron generates an interaction matrix, and feasible action priorities are computed via bilinear inner products, supporting adaptive decision-making.

Experimental results show that the proposed method achieves superior performance on standard benchmark datasets. Ablation studies confirm the contributions of both the job node embedding mechanism and the bilinear interaction mechanism, validating the framework's effectiveness in solving complex flexible job shop scheduling instances and offering methodological insights for intelligent scheduling research. Future work will extend the framework to dynamic scheduling scenarios—including machine unavailability, urgent order arrivals, and stochastic processing times—to enhance its applicability and resilience in real-world manufacturing settings.

References

1. Q. Zhang, A. Wang, J. Li, L. Zheng, J. Bao, and D. Zhang, "A Production-Logistics prediction method integrating Spatial-Temporal features in flexible production workshop for buffer allocation problem," *Computers & Industrial Engineering*, vol. 200, p. 110761, 2025.
2. M. Saidi-Mehrabad and P. Fattahi, "Flexible job shop scheduling with tabu search algorithms," *The International Journal of Advanced Manufacturing Technology*, vol. 32, nos. 5-6, pp. 563–570, 2007.

3. G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 56, no. 4, pp. 1309–1318, 2009.
4. R. Chen, B. Yang, S. Li, and S. Wang, "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 149, p. 106778, 2020.
5. B. A. Han and J. J. Yang, "Research on adaptive job shop scheduling problems based on dueling double DQN," *IEEE Access*, vol. 8, pp. 186474–186495, 2020.
6. X. Wu, X. Yan, D. Guan, and M. Wei, "A deep reinforcement learning model for dynamic job-shop scheduling problem with uncertain processing time," *Engineering Applications of Artificial Intelligence*, vol. 131, p. 107790, 2024.
7. K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, and L. Tang, "A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem," *Expert Systems with Applications*, vol. 205, p. 117796, 2022.
8. J. P. Huang, L. Gao, and X. Y. Li, "An end-to-end deep reinforcement learning method based on graph neural network for distributed job-shop scheduling problem," *Expert Systems with Applications*, vol. 238, p. 121756, 2024.
9. W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1600–1610, 2023.
10. E. Yuan, L. Wang, S. Cheng, S. Song, W. Fan, and Y. Li, "Solving flexible job shop scheduling problems via deep reinforcement learning," *Expert Systems with Applications*, vol. 245, p. 123019, 2024.
11. R. Wang, G. Wang, J. Sun, F. Deng, and J. Chen, "Flexible job shop scheduling via dual attention network-based reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 3, pp. 3091–3102, 2023.
12. K. H. Ho, J. Y. Cheng, J. H. Wu, F. Chiang, Y. C. Chen, Y. Y. Wu, and I. C. Wu, "Residual scheduling: A new reinforcement learning approach to solving job shop scheduling problem," *IEEE Access*, vol. 12, pp. 14703–14718, 2024.
13. C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1621–1632, 2020.
14. P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, no. 3, pp. 157–183, 1993.
15. J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *Operations-Research-Spektrum*, vol. 15, no. 4, pp. 205–215, 1994.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of Publisher and/or the editor(s). Publisher and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.