Learning Distributed Scheduling via LLM-Augmented Reinforcement Learning

Yun Liu[1], Yuqi Feng[1], Jiahao Fan[1], Shangce Gao[2], Yanan Sun[1]

[1]College of Computer Science, Sichuan University, Chengdu, China
[2]Faculty of Engineering, University of Toyama, Toyama, Japan
yliu@stu.scu.edu.cn, feng770623@gmail.com, fanjh@scu.edu.cn, gaosc@eng.u-toyama.ac.jp, ysun@scu.edu.cn

Abstract

The distributed flexible job-shop scheduling problem (DFJSP) aims to coordinate job execution across distributed factories to achieve production goals. Existing reinforcement learning (RL)-based scheduling algorithms have made processes in learning adaptive scheduling polices, but rely on shallow networks and simple handcrafted rewards. These designs limit global state reasoning and accurate credit assignment under sparse rewards, thereby hindering the balanced workload distribution and efficient policy learning. To address these limitations, we propose a Large Language Model (LLM)-augmented RL algorithm tailored for DFJSP by leveraging the contextual reasoning and prior knowledge of LLM. Specifically, we propose an LLM-driven factory assignment mechanism that encodes global factory states and job features into structured queries, enabling context-aware and effective coordination among factories. Furthermore, we design an LLM-informed reward model that encodes scheduling-aware semantics into multi-dimensional proxy rewards for precise credit assignment during training. Theoretical analysis establishes bounds on the reward approximation error and demonstrates that the designed factory assignment can effectively reduce global workload variance. Moreover, extensive experiments on two benchmarks (i.e., Hurink and Brandimarte) and simulation-based DFJSP instances of varying scales demonstrate that our algorithm outperforms state-of-the-art RL algorithms, achieving the average makespan improvement ranging from 0.61% up to 25.78%.

Code — https://anonymous.4open.science/r/LaRL-407B

Introduction

Scheduling is a fundamental process that involves managing, coordinating, and optimizing the execution of jobs and workload in a manufacturing system (Li et al. 2024). Among various scheduling problems, the distributed flexible job-shop scheduling problem (DFJSP) has attracted significant attention because it supports geographically distributed production, aligning with modern trends (Huang, Gao, and Li 2024a). In DFJSP, a set of jobs is assigned to multiple factories to optimize the desired objective (such as makespan or tardiness), after which each job is processed on a group of machines within the assigned factory based on the predefined operation sequence (Zhang et al. 2024). In practice, DFJSP is often subject to unexpected disturbances, especially in custom manufacturing companies, where flexible order placement leads to frequent new job arrivals (Huang, Gao, and Li 2024b). Therefore, it is critical for these companies to develop effective scheduling algorithms to ensure production efficiency in uncertain environments.

Existing scheduling algorithms can generally be divided into three categories: metaheuristics, heuristics, and reinforcement learning (RL)-based. Metaheuristics (Wang et al. 2025a) explore approximate solutions by iteratively evolving a population of solutions. Heuristics (Ito et al. 2022) allow obtaining feasible solutions by assigning priorities to jobs and factories based on specific criteria. Although the above algorithms can provide reasonable solutions, they typically rely on handcrafted rules and struggle to generalize across different scenarios. Recently, reinforcement learning (RL) has shown strong potential in eliminating handcrafted heuristics and enhancing adaptability in diverse scheduling scenarios (Zhang et al. 2020). RL-based scheduling algorithms can automatically learn a scheduling policy from experience to optimize long-term performance (Lei et al. 2023). One popular line is RL-based heuristic selection (Lei et al. 2024), which leverages RL to select among predefined heuristics based on environmental states. Another line adopts end-to-end learning (Huang, Gao, and Li 2024a), where the scheduling policy is directly learned from raw features without relying on handcrafted heuristics.

While these RL-based algorithms demonstrate promising results, they still face critical challenges in global coordination and reward design for solving DFJSP. First, **how to enable global coordination under limited state reasoning?** Most RL-based scheduling algorithms assume that factory assignment is handled by shallow policy networks (Huang, Gao, and Li 2024b) or simple heuristics (e.g., earliest available time and minimum transfer time) (Lei et al. 2024). These approaches lack sufficient state reasoning capacity to capture the global relationships between factories, resulting in poor coordination and load imbalance. Second, **how to achieve effective credit assignment under spare reward?** Existing algorithms often manually design scalar rewards based on the scheduling objectives (Lei et al. 2024). However, since these objectives typically depend on a com-

plete scheduling cycle, most intermediate actions receive no meaningful feedback during execution. This leads to ambiguous credit assignment, making it difficult for the agent to distinguish beneficial actions and hindering overall learning efficiency (Qu et al. 2025).

LLMs have shown strong capabilities in capturing global context and reasoning over structured inputs (Achiam et al. 2023), making them well-suited for effectively addressing the challenges of coordination and credit assignment in complex scheduling tasks. Motivated by these considerations, we propose an LLM-assisted RL algorithm, *LaRL*, to facilitate global coordination among factories and adaptive reward design for solving the DFJSP with new job arrivals. Specifically, we utilize LLM for assigning new jobs to factories and design an LLM-informed reward model to improve credit assignment in training agents of factories. The main contributions of our paper are summarized as follows:

1. We propose an LLM-driven factory assignment mechanism that leverages the contextual reasoning of LLMs to dynamically allocate newly arrived jobs based on global factory workload and job characteristics. This facilitates better global coordination and a more balanced workload of factories compared to shallow networks and heuristics.
2. We propose an LLM-informed reward model that exploits the prior knowledge of LLM to generate multifaceted proxy rewards for each action. This allows timely and informative feedback during execution, enabling more precise credit assignment under sparse reward compared to manual scalar rewards for scheduling.
3. We evaluate our algorithms on 167 DFJSP instances with six scales. The extensive experimental results demonstrate that our algorithm significantly outperforms state-of-the-art metaheuristics, heuristics, and RL algorithms, and demonstrates promising generalization to instances that are much larger than those used in training.

Related Work

Scheduling algorithms for DFJSP. Exist scheduling algorithms for DFJSP include metaheuristics, heuristics, and RL-based algorithms. Among them, metaheuristics, such as the memetic algorithm (Zhu et al. 2024), search for high-quality schedules by balancing global exploration and local exploitation. Over the last few years, various heuristics have been designed for DFJSP. For example, random search (Zabinsky et al. 2009) explores the solution space by uniformly sampling feasible solutions. Iterated greedy (Zhao et al. 2025) improves initial solutions through iterative destruction and reconstruction mechanisms. Dispatching rules (Huang, Gao, and Li 2024b) prioritize operations or machines based on predefined rules. However, these traditional methods often rely on handcrafted rules and lack adaptability in dynamic environments. With the development of RL, researchers have shifted to RL-based algorithms. Lei et al. (Lei et al. 2024) proposed a heuristic selection framework to choose among predefined heuristics based on the current state. Wang et al. (Wang et al. 2025b) introduced an end-to-end policy learning approach that directly maps raw environment features to scheduling actions. However, these RL-based algorithms still face challenges in achieving global coordination across factories and in credit assignment under sparse rewards. In this paper, we address these challenges via an end-to-end algorithm integrating global assignment and enhanced reward modeling.

LLMs-assisted decision-making. LLMs have recently emerged as powerful tools for complex decision-making tasks because of their advanced reasoning abilities and rich prior knowledge (Achiam et al. 2023). In particular, some recent studies have applied LLMs as planners to make decisions through APIs or predefined skills (Wang et al. 2024; Zhang et al. 2023). For example, Liu et al. (Liu et al. 2023) encode problem descriptions into a formal prompt to achieve long-horizon planning. Valmeekam et al. (Valmeekam et al. 2023) highlight the potential of LLMs in structured planning tasks. Beyond their planning capabilities, LLMs possess remarkable code generation ability that facilitates the automation of function design and decision (Jiang et al. 2024). Recent works have shown that LLMs can generate, debug, and optimize code snippets, significantly accelerating development cycles (Zhong, Wang, and Shang 2024). Inspired by these promising works, we propose leveraging the dual strengths of LLMs in planning and code generation to assist RL in solving DFJSP, aiming to improve effectiveness and learning efficiency in dynamic environments.

Problem Definition

In DFJSP with new job arrivals, there are n successively arriving jobs to be processed on l distributed factories, to optimize a scheduling objective (e.g., *makespan* in this paper). Each factory $F_f(0 \leq f < l)$ is equipped with M_f machines, and each job $J_i(1 \leq i \leq n)$ has a sequence of k operations. Each operation $O_{ij}(1 \leq j \leq k)$ is assigned to one of its candidate machines, with its processing time (p_{ij}) depending on the selected machine. Solving DFJSP involves three key tasks: 1) assigning each newly arrived job to a specific factory; 2) selecting a machine for each operation within the assigned factory; and 3) determining the processing sequence of operations on each machine. After the above tasks, we can derive a *schedule*, i.e., the start times (S_{ij}) of each operation and their corresponding machine assignments, such that the makespan $C_{max} = max_{ij}\{C_{ij} = S_{ij} + p_{ij}\}$ is minimized subject to all the constraints. In line with prior work (Lei et al. 2024), some assumptions are adopted as follows:

- The transfer time of jobs is neglected.
- All factories and machines are available at time zero.
- Each machine can only process one operation at a time.
- Each operation must be processed without interruption.
- Each operation cannot be started until its previous operation is successfully completed.

In summary, DFJSP aims to assign incoming jobs to distributed factories and schedule their operations on machines to minimize makespan. The dynamic and sequential nature of DFJSP makes it well-suited for RL frameworks, which motivates our formalization as a Markov decision process.

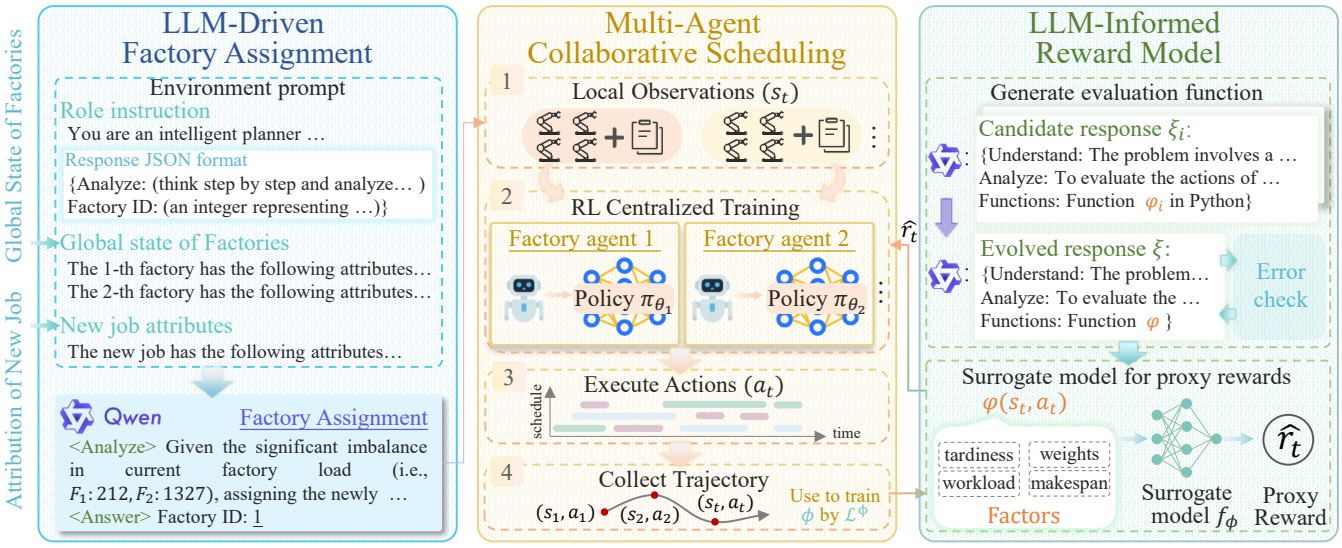

Figure 1: Overview of LaRL, which consists of three main components: (1) **LLM-driven factory assignment** assigns newly arrived jobs to factories by reasoning over the structured environment prompt consisting of the global factories state and new jobs attributions. (2) **Multi-agent collaborative scheduling** selects operation-machine pairs a_t within each factory based on the local observations o_t, and the trajectories are collected for training. (3) **LLM-informed reward model** computes proxy rewards via a learned surrogate model f_ϕ, which uses the evaluation function ϕ generated by LLM to decompose action contributions across multiple dimensions. Algorithm 1 in ***Technical Appendix B*** further details the relationships of the three components.

Proposed Algorithm

This section presents the proposed LLM-assisted RL algorithm, LaRL, for DFJSP with new job arrivals. We begin with an overview of LaRL, followed by its three components: LLM-driven factory assignment, multi-agent collaborative scheduling, and LLM-informed reward model.

Overview

As shown in Figure 1, LaRL consists of three components. First, LaRL uses LLM to guide job-factory assignment by encoding real-time states (e.g., load, availability, job features) into structured prompts, and infers the most suitable factory via semantic reasoning. Second, factory-specific multi-agent groups select operation-machine pairs based on local observations in a decentralized manner. Third, to address sparse rewards, we introduce an LLM-informed reward model. Specifically, the LLM defines a semantically interpretable evaluation function mapping observation-action pairs to multidimensional factors. We then train a surrogate model to estimate proxy rewards. This enables more accurate attribution of action contributions, facilitating stable and efficient policy learning. Our approach leverages the contextual understanding of LLMs to connect linguistic knowledge with symbolic decision-making, enhancing both effectiveness and generalization in DFJSP environments.

LLM-Driven Factory Assignment

To address the challenge of global coordination in DFJSP with new job arrivals, we propose an LLM-driven factory assignment mechanism that leverages the contextual reasoning capabilities and domain knowledge embedded in LLM. This mechanism determines the most suitable factory for each new job based on the global state, which is achieved by the following two steps 1) and 2).

1) *Environmental prompt*: The environment prompt serves to encode task-specific knowledge and contextual cues into a structured format, enabling the LLM to perform interpretable reasoning over the system state for decision-making. To this end, we construct the prompt P, defined as $P = Concat(R, G, A)$, where R denotes the role instruction assigning a role to LLM and describing the problem profile and objectives, $G = G_1, \ldots, G_l$ presents the global states of all l factories (e.g., workload, machine availability, and estimated delay ratio), A encodes all attributions of the new job (e.g., weights, due date, and expected time).

2) *LLM-based factory assignment*: Given the constructed prompt, the LLM can evaluate the relationship between the new job and each factory based on the encoded information, and generate factory assignments, where both the selected factory ID and its analysis are returned in JSON format. The assignment result is then passed to the downstream multi-agent collaborative scheduling. Our design not only enables more effective global coordination but also improves interpretability compared to shallow networks and heuristics. It is important to clarify that the LLM used in factory assignment is not fine-tuned during the training. This is because decisions are made infrequently (only upon new job arrivals), and the LLM already provides effective decisions through reasoning, as demonstrated in Section . Also, we provide an interpretability analysis in ***Technical Appendix I*** to demonstrate the ability of LLM in semantic reasoning.

Multi-Agent Collaborative Scheduling

To enable efficient scheduling within each factory, we formulate the factory-level scheduling problem as a multi-agent decision process. Each agent selects an operation-machine pair based on local observations using a designed policy, while coordination is ensured through centralized training. The local observation, policy, and actions are as follows.

1) *Local observation*: The status of each factory at decision step t is the input of the agent corresponding to factory F_f and defined as $o_t^f \in \mathbb{R}^{m \times d}$, where m is the number of available machines, d is the feature dimension. Each row in o_t^f represents a ready operation and consists of the processing time matrix P_t, the operation feature matrix F_t^o, and the machine feature matrix F_t^m, i.e., $o_t^f = [P_t, ||F_t^o||F_t^m]$. To address the varying number of ready operations, we fix the input size to the number of available machines, and apply zero-padding with a binary mask to filter out invalid rows. The details of the two feature matrices are provided in the ***Technical Appendix C***.

2) *Policy*: The policy π outputs a probability distribution over all actions to decide the executed actions. Scheduling decisions are highly frequent and involve structured operation–machine relations, which require fast and structure-aware inference. GATs naturally capture this bipartite structure while providing much lower latency than LLMs or other architectures (Veličković et al. 2018). Therefore, we adopt a GAT-based policy architecture, with all agents sharing it but maintaining factory-specific parameters. For each agent, given the local observation O_t^f, the policy computes the action distribution in three stages. First, GAT is used to encode the raw features of each machine M_m with its compatible ready operations $\mathcal{N}_t(M_m)$ into a v-dimensional embedding. By inputting P_t and F_t^m, GAT computes importance weights for compatible operations and aggregates their features to update the machine embeddings as Eq. (1):

$$e_t^m = \sigma(\alpha_{mm} W_M F_t^m + \sum_{O_{ij} \in \mathcal{N}_t(M_m)} \alpha_{ijm} W_O F_t^o) \quad (1)$$

where σ is an activation function, W_M and W_O are learnable matrices, α_{mm} and α_{ijm} are the attention coefficients representing the importance of machines to themselves and compatible operations.

Second, we use a multi-layer perceptron (MLP) to map the raw feature vectors of the ready operations into v-dimensional embeddings, as shown in Eq (2):

$$e_t^o = MLP_{\omega_0}(ELU[MLP_{\omega_1}(F_t^o)|| \sum_{m \in \mathcal{M}_t(O_{ij})} (e_t^m)]) \quad (2)$$

where $\mathcal{M}_t(O_{ij})$ is the candidate machines of O_{ij}, ω_0 and ω_1 are learnable parameters of the MLPs.

Finally, we derive the policy distribution over all feasible operation-machine pairs by concatenating the learned operation and machine embeddings $[\mathbf{e}_{ij}^o, \mathbf{e}_{ij}^c]$ and passing them through another MLP followed by a softmax layer in Eq. (3):

$$P(a_t^f, O_t^f) = softmax(MLP_{\omega_2}(e_t^o||e_t^m)) \quad (3)$$

where the MLP_{ω_2} with learnable parameters ω_2 consists of two hidden layers and Tanh activation.

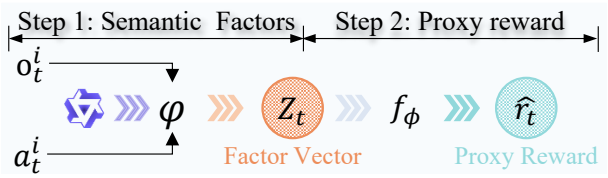

Figure 2: Pipeline of the LLM-informed reward model: (1) LLM generates an executable evaluation function φ that maps observation-action pairs (o_t^i, a_t^i) to factor vectors z_t; (2) a surrogate model f_ϕ predicts proxy rewards $\hat{r}_t$ from z_t.

3) *Action*: A valid operation-machine pair from the ready set corresponding to each factor F_f. Specifically, the action can be defined as $a_t^f = (O_{ij}, m_h)$, which indicates that the operation O_{ij} from job J_i is assigned to an idle machine $m_h \in M_f$. The action is selected from a masked probability distribution over the feasible operation-machine combinations, where infeasible actions (e.g., machines not in the candidate set) are masked out to ensure valid execution. Based on these decisions, we collect the joint trajectories $\tau = \{(s_t, a_t)\}_{t=1}^T$ for all factories, where $s_t = \{o_t^1, \ldots, o_t^l\}$ and $a_t = \{a_t^0, \ldots a_t^l\}$. These trajectories are used for centralized training of polices via Proximal Policy Optimization. More training details are in ***Technical Appendix E***.

LLM-Informed Reward Model

It is well known that key scheduling metrics such as tardiness and makespan can only be obtained after the entire schedule is completed. Therefore, designing rewards directly based on these metrics results in sparse feedback and hinders effective step-wise credit assignment, especially in large-scale DFJSP. To mitigate this limitation, we propose an LLM-informed reward model that leverages the prior knowledge and reasoning capabilities of LLMs. This model addresses two key challenges: 1) how to effectively ask the LLM to produce helpful reward signals that are reliable and consistent with symbolic in RL and DFJSP, and 2) how to use these signals to better assign credit to actions taken at different time steps. To this end, the proposed LLM-informed reward model first uses the LLM to generate an evaluation function that maps each observation-action pair to a structured semantic factor vector (Step 1 in Figure 2), and then trains a surrogate model to convert these factors into proxy rewards for training in RL (Step 2 in Figure 2). These two steps correspond to the two core components in the LLM-informed reward model, i.e., generating evaluation functions and training a surrogate model for proxy rewards, as illustrated in the third column of Figure 1.

1) *generating evaluation functions*: Inspired by previous work (Qu et al. 2025), we adopt a two-stage generation process consisting of LLM-based generation and self-evolution phases. In the generation phase, we first construct a structured prompt by encoding the role instruction, problem description, global scheduling state, and agent action formats, detailed in ***Technical Appendix A***. Then, the prompt is passed to LLM to produce z candidate responses $\{\xi_i, \ldots, \xi_z\}$, each of which involves an executable code of

an evaluation function φ_i. In the self-evolution phase, these functions are reorganized into a structured prompt. It guides the LLM to summarize a refined function φ that integrates the strengths of candidates while reducing redundancy and inconsistency, as shown in Eq. (4).

$$\varphi = LLM(problem, role, \varphi_1, \ldots, \varphi_z) \tag{4}$$

Furthermore, to ensure the executability of φ, we perform a preliminary error check by testing φ on random observation-action pairs. If any runtime errors occur, the corresponding error logs err are appended to the prompt to guide the LLM in refining the function again, as detailed in Eq. (5).

$$\varphi = LLM(problem, role, \varphi, err) \tag{5}$$

This two-stage process ensures that φ not only captures semantically meaningful aspects of agent actions, but is also syntactically executable and aligned with the scheduling objectives. Regard to this process, the function φ can take per-agent observation-action pair (o_t^i, a_t^i) and output a structured semantic factor vector $z_t = \varphi(o_t^i, a_t^i) = [z_t^1, \ldots, z_t^d]$.

2) *Surrogate model for proxy reward*: Considering φ is a symbolic and potentially non-differentiable function, build a surrogate model f_ϕ parameterized by ϕ based on the return decomposition (Efroni, Merlis, and Mannor 2021). This model can approximate the mapping from the observation-action pairs to scalar rewards. Specifically, the model estimates a proxy reward $\hat{r_t}$ from $\varphi(o_t^i, a_t^i)$ as $\hat{r_t} = f_\phi(\varphi(o_t^i, a_t^i))$. This per-agent proxy reward is used directly as the step-wise reward for policy optimization in RL, enabling finer-grained credit assignment in multi-agent settings. More details are in **Technical Appendix E**.

To align the proxy rewards with the true episodic returns $R(\tau)$ collected from trajectories τ, the surrogate model is trained by minimizing the loss function in Eq. (6).

$$\phi^* = \arg\min_\phi \mathbb{E}_{\tau\sim\pi}\left[\left(R(\tau) - \sum_{i=1}^{l}\sum_{t=1}^{T} f_\phi\big(\varphi(o_t^i, a_t^i)\big)\right)^2\right] \tag{6}$$

This design enables the surrogate to capture how agent-specific semantic factors contribute to the scheduling objective, thereby bridging LLM-derived semantics with RL rewards and improving learning efficiency in sparse-reward settings, especially in large-scale scheduling scenarios.

Theoretical Analysis

To justify the effectiveness of LaRL, we analyze how accurately the surrogate model f_ϕ approximates the true episodic return $R(\tau)$ through the per-agent proxy rewards derived from the symbolic evaluation $\varphi(o_t^i, a_t^i)$. We formalize the resulting decomposition error and provide an upper bound.

Theorem 1 (Reward Decomposition Bound). *Assume that the episodic return admits a linear decomposition over the LLM-generated factors, i.e., there exists f_ϕ^* such that $R(\tau) = \sum_{i=1}^{l}\sum_{t=1}^{T} f_\phi^*(\varphi(o_t^i, a_t^i))$. Let the surrogate estimator be $\hat{R}(\tau) = \sum_{i=1}^{l}\sum_{t=1}^{T} f_\phi(\varphi(o_t^i, a_t^i))$, and define the decomposition error as $\left\|R(\tau) - \hat{R}(\tau)\right\|_{A_k}$, where $A_k = \sum_{\tau\in\mathcal{D}} \varphi_\tau \varphi_\tau^\top + \lambda I$, with $\mathcal{D}$ denoting the set of k sampled trajectories. Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$, the following bound holds:*

$$\left\|R(\tau) - \hat{R}(\tau)\right\|_{A_k} \leq \sqrt{Dl\log\left(1 + \frac{kT^2 l}{\lambda\delta}\right)} + \sqrt{\lambda D} \tag{7}$$

where T is the episode length and $D = \dim(\varphi)$ is the factor dimension of the evaluation function.

Theorem 2 (LLM-driven Factory Assignment Improves Global workload Balance).
Let $\boldsymbol{L}_T = [L_T^1, \ldots, L_T^l]$ denote the cumulative workload over l factories after scheduling n jobs. Suppose each job J_i arrives from a stationary distribution and has a bounded processing time $p_t \in [0, p_{max}]$. If each job is assigned using an LLM-driven policy π_{LLM} with bounded assignment error ϵ, then the expected workload variance satisfies:

$$\mathbb{E}[Var(\mathbf{L}_T)] \leq \frac{C}{T} + \epsilon^3 \cdot p_{max}^2 \tag{8}$$

where C is a constant that represents the baseline workload variance caused by the job arrival distribution. The proofs of Theorems 1 and 2 are provided in ***Technical Appendix D***.

Experiments

Experimental Setup

1) Datasets. To evaluate LaRL, we conducted experiments on benchmark-based and simulation-based datasets. The benchmark datasets are derived from Hurink (Hurink, Jurisch, and Thole 1994) and Brandimarte (Brandimarte 1993) with two identical factories and varying jobs over $30, 50, 100$, following the convention (Zhang et al. 2024). The simulation-based datasets include 1,000/2,000/5,000 sequentially arriving jobs, following the practice (Zhang et al. 2024). Details are in ***Technical Appendix F.1***.

2) Peer competitors. We compare LaRL with six representative state-of-the-art algorithms from three categories. The first includes three popular heuristics: random search (RS) (Zabinsky et al. 2009), iterated greedy (PBIGA) (Zhao et al. 2025), and a dispatching rule (AR_SPT) (Huang, Gao, and Li 2024b). The second is a representative metaheuristic, i.e., RMA (Zhu et al. 2024). The third is two state-of-the-art RL-based algorithms: PPOS (Lei et al. 2024) and P-G (Wang et al. 2025b). More details are in ***Technical Appendix F.2***

3) Parameter Settings. The training settings follow (Li et al. 2024) with the batch size 128 and the initialized learning rate 1×10^{-4} (decayed by 0.96/epoch). The policy employs a GAT with single-head attention with ELU activation, and an output embedding dimension of eight. The surrogate reward model is implemented as a three-layer MLP with ReLU activation and a hidden size of 256. We use the public LLM, *Qwen-max*, for reasoning in LaRL. More details are in ***Technical Appendix F.3***.

4) Evaluation Criteria. We evaluate the performance using the average makespan (Mspan) over instances of each dataset. To assess load balance, we report the workload ratio (WR), defined as the ratio between the maximum and minimum total workloads across factories. Lower Mspan and WR closer to one indicate better performance.

Algorithm	DFJSP-30		DFJSP-50		DFJSP-100		DFJSP-1,000		DFJSP-2,000		DFJSP-5,000	
	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR
RS	2885.35(+)	1.975(+)	3536.59(+)	2.17(+)	6912.34(+)	1.66(+)	56241.32(+)	1.85(+)	127401.85(+)	1.95(+)	268174.60(+)	2.07(+)
PBIGA	2434.51(+)	1.32(+)	3506.24(+)	1.13(=)	6895.34(+)	1.42(+)	52889.34(+)	1.40(+)	119447.74(+)	1.69(+)	277748.52(+)	1.62(+)
AR_SPT	2696.47(+)	1.55(+)	3429.61(+)	1.50(+)	6599.55(+)	1.43(+)	53141.93(+)	1.36(+)	111066.90(+)	1.31(+)	267955.80(+)	1.96(+)
RMA	2422.35(+)	**1.03(=)**	3482.51(+)	1.31(+)	6382.62(+)	1.53(+)	53317.53(+)	1.48(+)	117374.21(+)	1.59(+)	277512.01(+)	1.69(+)
PPOS	2262.50(+)	1.79(+)	3229.15(+)	1.69(+)	6367.15(+)	1.55(+)	52994.87(+)	1.64(+)	111302.20(+)	1.53(+)	267729.40(+)	1.48(+)
P-G	**2096.24(+)**	1.41(+)	**3176.34(=)**	1.36(+)	6266.21(+)	1.36(=)	52581.76(=)	1.20(+)	112897.36(+)	1.65(+)	267684.27(+)	1.41(+)
LaRL (Ours)	2141.00	1.15	3180.22(=)	**1.12**	**6254.50**	**1.31**	**52241.50**	**1.07**	**107374.00**	**1.03**	**266537.53**	**1.16**

Table 1: Comparative study on different algorithms. We report the *average* makespan (Mspan) and workload ratio (WR) on DFJSP instances of various scales. The best performances are highlighted in bold. Symbols '+', '=', and '–' indicate that our proposed algorithm LaRL is significantly better, equivalent, or worse than the peers via the Wilcoxon rank-sum test ($p < 0.05$).

Scale (f-m-n)	(3-10-30)		(3-10-50)		(3-10-100)		(3-10-1,000)		(3-10-2,000)		(3-10-5,000)	
Algorithm	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR
RS	2499.27(+)	2.67(+)	3036.62(+)	2.19(+)	6289.12(+)	3.32(+)	59253.71(+)	2.69(+)	119471.21(+)	2.91(+)	268174.60(+)	3.71(+)
PBIGA	2362.41(+)	1.45(+)	2911.34(+)	1.28(=)	6233.52(+)	1.46(+)	58756.86(+)	1.54(+)	120733.74(+)	1.34(+)	278450.33(+)	3.27(+)
AR_SPT	2233.16(+)	2.15(+)	2925.52(+)	1.42(+)	5994.70(+)	1.24(+)	56886.37(+)	1.62(+)	119281.73(+)	1.47(+)	269655.73(+)	2.76(+)
RMA	2491.17(+)	1.58(+)	2999.46(+)	1.13(+)	6187.05(+)	1.53(+)	57898.27(+)	1.59(+)	121241.17(+)	1.39(+)	283474.71(+)	3.20(+)
PPOS	2143.27(+)	1.56(+)	2950.00(+)	1.38(+)	6050.71(+)	1.32(+)	57083.15(+)	1.47(+)	110362.09(+)	1.60(+)	269972.53(+)	2.09(+)
P-G	2096.24(+)	1.68(+)	2927.71(=)	1.29(+)	5996.05(+)	1.34(=)	56483.62(+)	1.37(+)	110210.47(+)	1.51(+)	269547.63(+)	1.83(+)
LaRL (Ours)	**2048.50**	**1.22**	**2841.86**	**1.04**	**5820.06**	**1.17**	**55205.66**	**1.16**	**109648.66**	**1.12**	**268870.06**	**1.23**

Table 2: Comparative study on the instances with three factories. We report the *average* makespan (Mspan) and workload ratio (WR) on six instance sizes, where (f-m-n) means there are f factories, m machines per factory, and n jobs. The best performances are highlighted in bold. Statistical significance follows the same notation as Table 1.

Results and Analysis

(1) Comparative Study

In this section, we evaluate LaRL against peer competitors on DFJSP instances with varying scales. The evaluation is based on the *average* makespan (Mspan) and workload ratio (WR) and statistical significance is assessed using the Wilcoxon rank-sum test ($p < 0.05$), where symbols '+', '=', and '–' indicate that LaRL performs significantly better, equivalent, or worse than the competitors.

Performance on instances with two factories. Table 1 presents the comparative results of LaRL and six representative scheduling algorithms on six datasets with varying scales. LaRL achieves the best WR on all datasets and outperforms all baselines in makespan on four of six instances, demonstrating its strength in global coordination and workload balancing. In terms of makespan, on small-scale instances (DFJSP-30/50/100), LaRL performs on par with or better than P-G, and consistently surpasses heuristic and metaheuristic baselines. On large-scale instances (DFJSP-1,000/2,000/5,000), it achieves the lowest makespan and WR across the board. In terms of WR, LaRL reduces WR to near 1.0 on all instances, with 1.16 on DFJSP-5,000, while others exceed 1.4, indicating superior workload balancing. All improvements are statistically significant under the Wilcoxon signed-rank test ($p < 0.05$), indicating that LaRL consistently outperforms existing algorithms in both makespan and workload balancing.

Performance on instances with more factories. To further evaluate the generalization of LaRL under different factory configurations, we evaluate LaRL on scenarios with three factories. Table 2 reports the average makespan and workload rate of all algorithms. Across all problem scales, LaRL consistently outperforms competitors in minimizing makespan, demonstrating its generalization ability when deployed in environments with more distributed factories. Notably, LaRL maintains balanced workload distribution among factories, as indicated by the reported workload rates, which remain close across factories. In contrast, peer competitors often suffer from workload skew, especially in large-scale instances (e.g., 2,000 and 5,000 jobs). These results confirm that the reasoning-based assignment and scheduling mechanism can effectively maintain both scheduling quality and workload balance in increasingly complex settings.

Robustness to the frequency of job arrivals. To evaluate the robustness of LaRL under different frequencies of job arrivals, we vary the utilization level from the default high-utilization setting (0.95) to a moderate level (0.85), which is commonly adopted in the scheduling literature (Zhang, Mei, and Zhang 2023). Experiments are conducted on instances with two factories, each equipped with 10 machines, and varying job sizes $1,000, 2,000, 5,000$. As summarized in Table 3, LaRL consistently achieves lower makespan across all scales, with the most pronounced gains at 5,000 jobs. These findings confirm that LaRL is not only effective in high-pressure environments but also maintains superior when the frequency of job arrivals decreases.

(2) Ablation Study

Contribution of the proposed LLM-driven factory assignment. To evaluate the contribution of the LLM-driven factory assignment, we replace it with two alternative strate-

Method	DFJSP-30		DFJSP-50		DFJSP-100		DFJSP-1,000		DFJSP-2,000		DFJSP-5,000	
	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR	Mspan	WR
RS	2215.73(+)	1.69(+)	3855.36(+)	1.35(+)	6062.15(+)	1.81(+)	60280.38(+)	2.91(+)	132015.47(+)	3.26(+)	320781.53(+)	2.26(+)
PBIGA	2090.34(+)	1.49(+)	3650.06(+)	1.14(=)	5801.75(+)	1.63(+)	61259.82(+)	2.41(+)	132154.22(+)	2.16(+)	325649.76(+)	1.98(+)
AR_SPT	2696.47(+)	1.55(+)	3732.71(+)	1.30(+)	5948.28(+)	1.72(+)	60745.58(+)	1.96(+)	129916.15(+)	1.96(+)	319984.66(+)	1.87(+)
RMA	2178.26(+)	1.41(=)	3415.29(+)	1.21(+)	5868.10(+)	1.51(+)	62046.62(+)	2.33(+)	132516.37(+)	2.01(+)	328975.61(+)	2.09(+)
PPOS	2078.31(+)	1.51(+)	3370.29(+)	1.27(+)	5810.53(+)	1.35(+)	59251.58(+)	1.45(+)	165368.73(+)	1.59(+)	310786.91(+)	1.76(+)
P-G	**2006.73(-)**	**1.02(-)**	3227.20(+)	**1.15(=)**	**5736.61(-)**	1.31(=)	59259.95(=)	1.49(+)	122259.23(+)	1.76(+)	308276.36(+)	1.69(+)
LaRL (Ours)	2026.56	1.09	**3180.66(=)**	1.17	5764.36	**1.28**	**58986.26**	**1.18**	**120126.65**	**1.11**	**293905.67**	**1.21**

Table 3: Comparative study on different algorithms. We report the *average* makespan and WR of factories in the scenario with a utilization level of 0.85. The best performances are in bold. '+', '=', and '−' indicate statistically better, equivalent, and worse performance of LaRL compared to others (Wilcoxon test, $p < 0.05$).

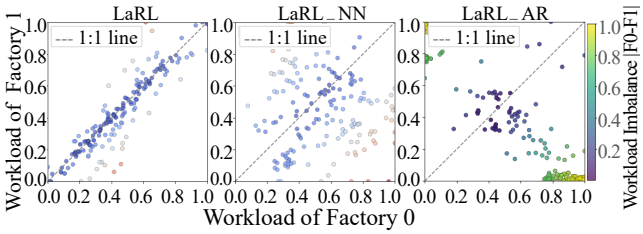

Figure 3: Comparison of workload of two factories for three algorithms (LaRL, LaRL_NN, and LaRL_AR). Each subfigure shows the normalized workload assigned to both factories on all instances, with the 1:1 diagonal line (gray dashed) indicating perfect workload balance. The color intensity reflects the absolute difference of workload ($|F0-F1|$), where darker colors indicate smaller differences between factories.

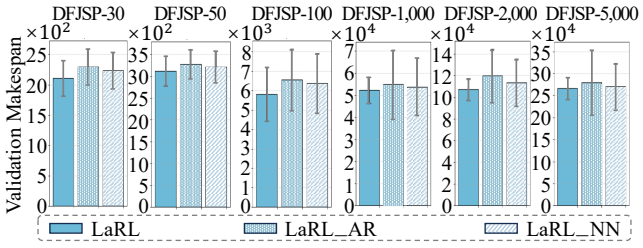

Figure 4: Comparison of makespan between LaRL with two variants on six datasets. LaRL outperforms its variants in both average performance and robustness, highlighting the effectiveness of the LLM-driven factory assignment.

gies: (1) a heuristic-based method using the classic AR rule (Huang, Gao, and Li 2024b), and (2) a learned neural network that maps global factory states and job attributes to assignment decisions (Lei et al. 2024). These two variants are referred to as LaRL_AR and LaRL_NN, respectively.

Figure 3 visualizes the factory-level load distribution using scatter plots. Each subfigure shows the normalized workload assigned to both factories on all instances, with the 1:1 diagonal line (gray dashed) indicating perfect workload balance. The color intensity reflects the absolute difference of workload ($|F0 - F1|$), where darker colors indicate smaller differences between factories. LaRL exhibits

Testing Datasets	LaRL		LaRL_HR	
	Mspan	WR	Mspan	WR
DFJSP-30	2141.00	1.15	2198.72(+)	1.27(+)
DFJSP-50	3180.22	1.12	3210.64(+)	1.16(+)
DFJSP-100	6254.50	1.31	6379.46(+)	1.29(=)
DFJSP-1,000	52241.50	1.07	52976.96(+)	1.13(+)
DFJSP-2,000	107374.00	1.03	112951.21(+)	1.09(+)
DFJSP-5,000	266537.53	1.16	269792.68(+)	1.23(+)

Table 4: Comparison of average makespan and WR between LaRL and LaRL_HR over six datasets with different scales. Statistical significance follows the same notation as Table 1.

the most concentrated distribution along the diagonal line, reflecting highly superior balance. In contrast, LaRL-AR shows frequent skewed allocations due to static rules, while LaRL-NN shows moderate imbalance with less consistency. These results highlight the advantage of LLM-based assignment in leveraging global context and semantics to coordinate job allocation more effectively. Figure 4 shows that LaRL achieves consistently lower average makespan and variance across six datasets, indicating both superior performance and robustness. Notably, as the scale increases, the variance of LaRL remains significantly lower, highlighting its stability under complex settings. This suggests that the LLM can integrate diverse features and reason contextually, while rule-based or shallow models often rely on limited criteria, yielding instability under large-scale instances.

Contribution of the proposed LLM-informed reward model. To evaluate the contribution of the LLM-informed reward model, we conduct an ablation study by replacing it with a widely used handcrafted reward function (Lei et al. 2024), denoted as LaRL-HR. Both variants remain the same except the reward function.

Table 4 presents the comparative results between LaRL and its variant LaRL_HR. Across all six datasets with increasing problem scales, LaRL consistently achieves lower makespan and better or comparable WR on all instances. In particular, the relative advantage of LaRL is more evident in larger-scale settings. Figure 5 illustrates the normalized episodic return curves under two different shop utilization levels (0.95 and 0.85). In both settings, LaRL consistently converges faster and exhibits more stable learning with narrower shaded regions, indicating improved learning

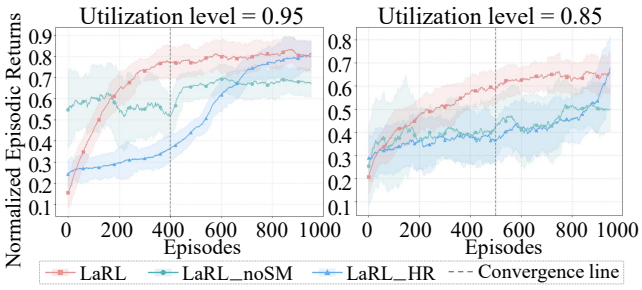

Figure 5: Episodic return curves of LaRL, LaRL_HR, and LaRL_noSM under two utilization levels. The solid lines denote smoothed normalized returns over training episodes, and the shaded regions indicate the standard deviation.

Datasets	DFJSP-30		DFJSP-50		DFJSP-100	
Algorithm	Mspan	WR	Mspan	WR	Mspan	WR
Deepseek-V3	2152.00	1.16	3190.88	1.12	6341.75	1.26
ChatGPT-3.5	2092.60	1.05	3079.00	1.13	6217.67	1.21
Qwen-max	2141.00	1.15	3180.22	1.12	6254.50	1.31

Table 5: Comparison of LaRL performance using different LLMs (i.e., Qwen-max, DeepSeek-V3, and ChatGPT-3.5) on the benchmark-based datasets.

efficiency and robustness, especially under a higher utilization level (i.e., more frequent job arrivals). This validates that the LLM-guided reward model not only enhances credit assignment under sparse feedback but also adapts well to varying scheduling intensities. This is because its proxy reward is learned from LLM-derived semantic factors rather than fixed handcrafted form, yielding a data-driven signal that aligns more closely with the objective.

Necessity of surrogate model. We introduce an additional ablation variant, LaRL_noSM, which removes the surrogate model and calculates the reward by summarizing the semantic factors. As shown in Figure 5, LaRL_noSM performs worse than LaRL, confirming that the surrogate model is essential for translating LLM-generated factors into effective step-wise rewards that enable accurate credit assignment.

Impact of LLM choice. To evaluate the impact of different LLMs on the performance of LaRL, we compare LaRL using Qwen-max (default), DeepSeek-V3, and ChatGPT-3.5 on the benchmark-based datasets. As shown in Table 5, although all variants benefit from LLM, stronger models such as ChatGPT-3.5 achieve lower average makespan and balanced workload. Qwen-max, adopted as the default because of its strong open-source accessibility and stable reasoning quality, achieves competitive performance across all settings. These results suggest that LaRL is robust to LLM choice and can further improve when equipped with more powerful models, highlighting its potential as a scalable framework for practical deployment.

Impact of GNN choice. Figure 6 presents the comparison between LaRL (GAT) and its two variants using GCN and GIN. Across all scales, LaRL-GCN consistently underperforms GAT, confirming that simple message passing struggles to capture complex operation-machine relations. LaRL-GIN achieves marginal gains over GAT on DFJSP-100 and DFJSP-2000, but falls behind on the remaining instances, especially as problem size increases. These results highlight that the attention mechanism of GAT is more suitable for distributed scheduling tasks, as it dynamically weighs operation-machine compatibility and supports selective information aggregation under large-scale environments.

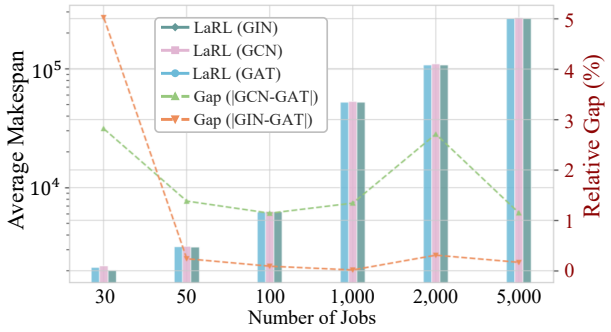

Figure 6: Comparison of LaRL variants with three GNN backbones (GCN, GIN, GAT) on instances of varying scale. The left Y-axis (log scale) shows the average makespan; the right Y-axis shows the relative gap (%) to LaRL(GAT).

Analysis of Time Complexity. Let n denote the number of jobs, m the number of machines per factory, l the number of factories, d the average number of ready operations per machine, and v the embedding dimension. Heuristics run in $\mathcal{O}(nlogn)$ to $\mathcal{O}(nm)$, while the metaheuristic RMA has a complexity of $\mathcal{O}(nmg)$ with g as the population size. RL-based baselines typically involve per-step policy inference with complexity $\mathcal{O}(v^2)$ and $\mathcal{O}(lvn^2)$. LaRL introduces additional cost from the LLM-based factory assignment and GAT-based multi-agent scheduling, leading to an overall complexity of $\mathcal{O}(T_{LLM}+ldv^2)$. Despite this, LaRL remains highly efficient in practice, as the LLM is only invoked when new jobs arrive, and the scheduling decisions are made locally within each factory. Additionally, ***Technical Appendix H*** provides a dedicated analysis of computational cost to quantify the latency from LLM calls, while ***Technical Appendix J*** further discusses the practical feasibility of LaRL in real scheduling environments.

More Experiments Are in the Appendix. More experiments include evaluations on fewer machines (i.e., five machines per factory) (***Technical Appendix G***) and interpretability of factory assignment (***Technical Appendix I***).

Conclusion

This paper addresses two core challenges: limited effective global coordination and the difficulty of credit assignment under sparse rewards. To this end, we propose an LLM-assisted RL algorithm, LaRL, which leverages the reasoning capability and domain knowledge of LLMs to guide factory assignment and construct multi-factor rewards. LaRL improves coordinated scheduling and efficient training, yielding better makespan and workload balance across scales.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Brandimarte, P. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3): 157–183.

Efroni, Y.; Merlis, N.; and Mannor, S. 2021. Reinforcement learning with trajectory feedback. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 7288–7295.

Huang, J.-P.; Gao, L.; and Li, X.-Y. 2024a. An end-to-end deep reinforcement learning method based on graph neural network for distributed job-shop scheduling problem. *Expert Systems with Applications*, 238: 121756.

Huang, J.-P.; Gao, L.; and Li, X.-Y. 2024b. A hierarchical multi-action deep reinforcement learning method for dynamic distributed job-shop scheduling problem with job arrivals. *IEEE Transactions on Automation Science and Engineering*.

Hurink, J.; Jurisch, B.; and Thole, M. 1994. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15: 205–215.

Ito, S.; Kanahara, K.; Oda, T.; and Katayama, K. 2022. An Extended NEH based Method for Permutation Flowshop Scheduling Problem. In *Proceedings of the 10th International Conference on Computer and Communications Management*, ICCCM '22, 252–256. New York, NY, USA: Association for Computing Machinery. ISBN 9781450396349.

Jiang, J.; Wang, F.; Shen, J.; Kim, S.; and Kim, S. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Lei, K.; Guo, P.; Wang, Y.; Zhang, J.; Meng, X.; and Qian, L. 2023. Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning. *IEEE Transactions on Industrial Informatics*, 20(1): 1007–1018.

Lei, Y.; Deng, Q.; Liao, M.; and Gao, S. 2024. Deep reinforcement learning for dynamic distributed job shop scheduling problem with transfers. *Expert Systems with Applications*, 251: 123970.

Li, L.; Liang, S.; Zhu, Z.; Ding, C.; Zha, H.; and Wu, B. 2024. Learning to optimize permutation flow shop scheduling via graph-based imitation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20185–20193.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Qu, Y.; Jiang, Y.; Wang, B.; Mao, Y.; Wang, C.; Liu, C.; and Ji, X. 2025. Latent reward: Llm-empowered credit assignment in episodic reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 20095–20103.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the Planning Abilities of Large Language Models - A Critical Investigation. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 75993–76005. Curran Associates, Inc.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Wang, C.; Wei, M.; Liu, Q.; Zhang, X.; and Li, X. 2025a. An improved adaptive hybrid algorithm for solving distributed flexible job shop scheduling problem. *Swarm and Evolutionary Computation*, 94: 101873.

Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6): 186345.

Wang, R.; Jing, Y.; Gu, C.; He, S.; and Chen, J. 2025b. End-to-End Multitarget Flexible Job Shop Scheduling With Deep Reinforcement Learning. *IEEE Internet of Things Journal*, 12(4): 4420–4434.

Zabinsky, Z. B.; et al. 2009. Random search algorithms. *Department of Industrial and Systems Engineering, University of Washington, USA*, 34.

Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P. S.; and Chi, X. 2020. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33: 1621–1632.

Zhang, D.; Chen, L.; Zhang, S.; Xu, H.; Zhao, Z.; and Yu, K. 2023. Large language models are semi-parametric reinforcement learning agents. *Advances in Neural Information Processing Systems*, 36: 78227–78239.

Zhang, F.; Mei, Y.; and Zhang, M. 2023. An investigation of terminal settings on multitask multi-objective dynamic flexible job shop scheduling with genetic programming. In *Proceedings of the companion conference on genetic and evolutionary computation*, 259–262.

Zhang, Z.; Fu, Y.; Gao, K.; Pan, Q.; and Huang, M. 2024. A learning-driven multi-objective cooperative artificial bee colony algorithm for distributed flexible job shop scheduling problems with preventive maintenance and transportation operations. *Computers & Industrial Engineering*, 196: 110484.

Zhao, F.; Du, Y.; Zhuang, C.; Wang, L.; and Yu, Y. 2025. An Iterative Greedy Algorithm for Solving a Multiobjective Distributed Assembly Flexible Job Shop Scheduling Problem With Fuzzy Processing Time. *IEEE Transactions on Cybernetics*, 55(5): 2302–2315.

Zhong, L.; Wang, Z.; and Shang, J. 2024. Debug like a Human: A Large Language Model Debugger via Verifying Runtime Execution Step by Step. In *Findings of the Association for Computational Linguistics ACL 2024*, 851–870.

Zhu, N.; Gong, G.; Lu, D.; Huang, D.; Peng, N.; and Qi, H. 2024. An effective reformative memetic algorithm for distributed flexible job-shop scheduling problem with order cancellation. *Expert Systems with Applications*, 237: 121205.