Harmonizing Real-Time Constraints and Long-Horizon Reasoning: An Asynchronous Agentic Framework for Dynamic Scheduling

Shijie Cao[1,2], **Yuan Yuan**[1,3,4*] and **Jing Liu**[2,5,6]
[1]School of Computer Science and Engineering, Beihang University, Beijing 100191, China
[2]Shenzhen Loop Area Institute, Shenzhen, China
[3]Qingdao Research Institute, Beihang University
[4]Hangzhou Innovation Institute, Beihang University
[5]School of Artificial Intelligence, Xidian University, Xi'an 710071, Shaanxi, China
[6]Guangzhou Institute of Technology, Xidian University, Guangzhou 510555, Guangdong, China
{cls1277, yuan21}@buaa.edu.cn, neouma@mail.xidian.edu.cn

Abstract

The Dynamic Flexible Job Shop Scheduling Problem (DFJSP) necessitates a trade-off between instant reaction to stochastic disturbances and global optimization of production goals. Conventional priority rules are insufficiently flexible to handle complex disruptions, whereas learning-based approaches often compromise interpretability or fail to generalize across problem scales. Although Large Language Models (LLMs) offer advanced reasoning capabilities to bridge this gap, their substantial inference latency is incompatible with the millisecond-level decision cycles of industrial control systems. To resolve this conflict, we introduce RACE-Sched, an asynchronous agent-based framework that decouples policy execution from logical reasoning via a dual-stream architecture. The Reactive Stream executes low-latency symbolic heuristics to enable real-time dispatching, while the parallel Deliberative Stream leverages an LLM to synthesize, validate, and evolve these rules. Candidate rules undergo rigorous testing in a sandbox and are deployed via atomic updates, ensuring safety without blocking the control loop. Additionally, a semantic rule repository indexes validated heuristics for retrieval-based initialization which enhances transferability across problem scales. Extensive evaluations on GEN-Bench, MK-Bench, and JMS-Bench demonstrate that RACE-Sched outperforms leading Deep Reinforcement Learning and other LLM-based baselines. This approach harmonizes real-time constraints with long-horizon reasoning to achieve superior solution quality and robust adaptation to dynamic events.

1 Introduction

The Dynamic Flexible Job Shop Scheduling Problem (DFJSP) is a core online decision problem in modern manufacturing, where each ready operation must be assigned to a feasible heterogeneous machine under routing flexibility and precedence constraints [Xu *et al.*, 2025]. When machine failures, urgent job arrivals, or processing-time variations change the shop-floor state, delayed dispatching decisions can block available machines and propagate idle time through subsequent operations.

Data-driven dispatching policy learning is achievable with Deep Reinforcement Learning (DRL), but interpreting its decisions poses significant challenges and these decisions may be vulnerable to variations in training details [Zhang *et al.*, 2020]. Priority Dispatching Rules (PDRs) are easy to execute and inspect, yet their rigid structures often fail to adapt when the bottleneck shifts [Holthaus and Rajendran, 2000]. By generating executable rules with LLMs, Code as Policy (CaP) improves the transparency of scheduling decisions [Liang *et al.*, 2023]. One remaining obstacle is response latency: making LLM calls during the control loop operation is too slow to satisfy the real-time demands of scheduling systems.

The fundamental conflict lies in the temporal granularity between reasoning and execution. Advanced reasoning models such as GPT-4 require several seconds to process context and generate code [Hurst *et al.*, 2024]. In contrast, dynamic scheduling decisions in a high-speed production line must be made within milliseconds to avoid blocking machine operations [Singh *et al.*, 2026; Ouelhadj and Petrovic, 2009]. Simple integration strategies that pause the production line to await external reasoning result in unacceptable throughput losses. This leads to a frequency mismatch where the slow cognitive cycle of the LLM cannot synchronize with the fast physical cycle of the manufacturing floor [Karami *et al.*, 2025]. Consequently, current approaches are forced to compromise by either relying on pre-generated static rules that lack adaptability or employing smaller and less capable models that sacrifice reasoning quality for speed [Shah *et al.*, 2026].

Furthermore, existing hybrid frameworks typically separate learning from execution in a manner that precludes continuous online adaptation. Most methodologies rely on an offline training phase, where scheduling rules are derived from historical data [Priore *et al.*, 2014]. Once deployed, these

*Corresponding author.

rules remain static and cannot respond to unseen disturbances such as sudden machine breakdowns or drastic shifts in order priority. Although some adaptive systems attempt to periodically retrain policies, they often require synchronous interactions that disrupt the ongoing scheduling process. A key limitation in current research is the lack of mechanisms that allow for the safe and asynchronous evolution of control logic during system operations. An ideal scheduling system should enable the control logic to improve continuously without impeding the real-time responsiveness of physical machines.

While inspired by the asynchronous dual-stream concept [Chen *et al.*, 2026], directly applying existing embodied agents to industrial scheduling is hindered by the requirement for interpretability and strict safety guarantees. Rather than treating LLM-based code generation, simulation validation, and iterative refinement as new ingredients, we focus on how these mechanisms can be safely integrated into a running DFJSP control loop. We introduce RACE-Sched[1], a framework where the Reactive Stream runs low-latency symbolic heuristics, while the Deliberative Stream leverages an LLM to analyze summary statistics from a sliding window of recent decisions, generate candidate Python heuristics, and test them in a sandbox that replays representative instances under the same constraints. Only candidates that meet predefined acceptance criteria are promoted. The Reactive Stream does not lie on the critical path: it never waits for deliberation, and updates are executed through an atomic pointer swap of the active rule set.

We also maintain a rule repository that indexes validated heuristics together with lightweight instance-level metadata, such as numbers of jobs and machines. The Deliberative Stream retrieves the most similar rules to warm-start candidate generation and sandbox evaluation, which helps shorten prompts and enhance robustness across problem scales.

Our contributions are as follows.

- An asynchronous symbolic evolution framework adapted for industrial constraints, which decouples real-time dispatching from LLM-driven heuristic synthesis via a safe "code-as-policy" mechanism, ensuring the control loop remains responsive.
- A sandbox-based validation and safe deployment mechanism, where candidate heuristics are evaluated offline through constrained replay, and promoted only if they meet predefined acceptance criteria, followed by an atomic pointer swap of the active rule set.
- A rule repository that indexes validated heuristics with lightweight instance-level metadata, enabling warm-start retrieval and improving transferability across different benchmarks and problem scales.
- An extensive evaluation on GEN-Bench, MK-Bench, and JMS-Bench, including a machine-failure stress test, showing that our approach achieves higher solution quality and faster adaptation than competitive DRL baselines and direct LLM control.

[1]To facilitate reproducibility, our code is available at https://github.com/cls1277/RACE-Sched

2 Related Work

Heuristics and Evolutionary Scheduling. Priority Dispatching Rules remain the industrial standard for dynamic scheduling due to their minimal computational cost of constant time complexity and ease of interpretation [Holthaus and Rajendran, 2000]. However, fixed rules such as Shortest Processing Time and Most Work Remaining are inherently myopic. They rely on local buffer status and often fail to maintain performance when system bottlenecks shift due to stochastic disturbances. To automate the design of dispatching policies, Genetic Programming (GP) has been widely applied to evolve symbolic priority functions that combine production attributes into complex rules [Mei *et al.*, 2016]. While advances such as surrogate-assisted GP [Mei *et al.*, 2016] and multitask evolution [Zhang *et al.*, 2023] have improved convergence speed, the evolutionary process remains computationally expensive and offline. This limitation prevents GP from achieving the rapid online adaptation required to handle sudden machine failures or urgent order insertions in real-time environments.

DRL for Dynamic Scheduling. To capture complex system dynamics, recent research has shifted toward DRL by modeling scheduling as a Markov Decision Process [Xu *et al.*, 2025]. State representation has evolved from simple feature vectors to Graph Neural Networks that encode the non-Euclidean topology of job shops [Zhang *et al.*, 2025]. The Dual Attention Network [Wang *et al.*, 2024b] established a benchmark by jointly attending to operation precedence and machine contention. Specialized architectures have also been developed for specific disruptions such as IDDQN [Wu *et al.*, 2025] for robustness against machine breakdowns and hierarchical frameworks such as HMPSAC [Ding *et al.*, 2025] for multi-objective trade-offs. Despite these gains, DRL faces critical deployment barriers. The resulting black-box neural policies lack the interpretability required for safety-critical manufacturing [Li *et al.*, 2026] and often exhibit poor generalization when transferring between different problem scales.

LLM Reasoning and Code-as-Policy. LLMs offer a promising avenue to address the interpretability and reasoning gaps in DRL. While techniques such as Chain-of-Thought [Wei *et al.*, 2022] and Tree of Thoughts [Yao *et al.*, 2023] enhance logical planning, the high latency of inference creates a temporal mismatch with the millisecond-level response requirements of industrial control. To address this discrepancy, the CaP paradigm proposes generating executable programs rather than direct text actions [Liang *et al.*, 2023]. This approach generates fast and interpretable code as validated by open-ended agents such as Voyager [Wang *et al.*, 2024a]. In the scheduling domain, ReflecSched [Cao and Yuan, 2025] recently utilized LLMs to generate hierarchical reflections for guidance. Prior LLM reflection and CaP scheduling methods mainly focus on synthesizing or refining executable policies, while hybrid offline-online heuristic evolution and safe update schemes primarily address policy improvement and deployment risk as separate stages. RACE-Sched instead focuses on the control-integration problem: it confines slow LLM deliberation, sandbox validation, and candidate policy improvement to the background, while the

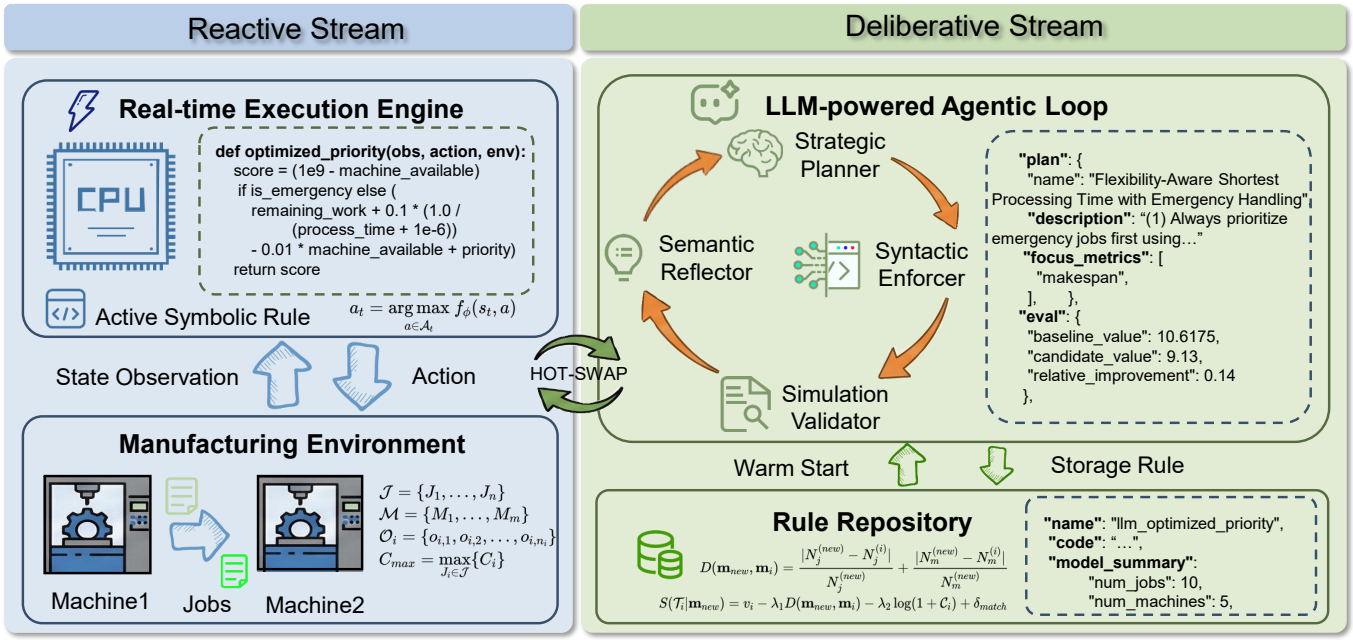

Figure 1: Overview of RACE-Sched. The Reactive Stream executes an active symbolic rule for real-time scheduling, while the Deliberative Stream runs an LLM-driven loop with constrained code generation and sandbox evaluation to produce validated rule updates. Validated heuristics are stored in a rule repository for warm-start retrieval and deployed to the control loop via hot-swap.

online scheduler executes only the currently validated symbolic rule and receives accepted updates through atomic hot-swap.

3 Problem Formulation and Preliminaries

We consider the DFJSP, which entails the assignment of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ to a set of m heterogeneous machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job $J_i \in \mathcal{J}$ comprises a strictly ordered sequence of n_i operations $\mathcal{O}_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,n_i}\}$. Routing flexibility permits each operation $o_{i,j}$ to be executed on any machine within an eligible subset $\mathcal{M}_{i,j} \subseteq \mathcal{M}$, where $p_{i,j,k}$ denotes the deterministic processing time on machine $M_k \in \mathcal{M}_{i,j}$ [Cao *et al.*, 2023].

The system state at time τ comprises the progress of active jobs, machine availability, and the backlog of pending operations. Temporal evolution is driven by asynchronous stochastic disturbances that induce discrete state transitions, including non-deterministic job arrivals and machine failure-recovery cycles. Under a preempt-resume scheduling policy, interrupted operations remain suspended and are resumed from the point of interruption upon the restoration of machine capacity. The objective is to minimize the expected makespan $C_{\max}$, where C_i denotes the completion time of job J_i and $C_{\max} = \max_{J_i \in \mathcal{J}} C_i$, subject to real-time decision constraints in a dynamic manufacturing environment.

We formulate this task as a sequential decision-making process where a policy π maps system states to operation-machine assignments at each decision epoch t, while satisfying precedence and availability constraints. In this stochastic setting, performance optimization requires strategic reasoning to address complex operational tradeoffs, such as the proactive allocation of machine capacity to alleviate anticipated bottlenecks.

4 Methodology

A detailed workflow diagram of RACE-Sched is provided in Figure 1. The Reactive Stream runs a symbolic priority rule inside the control loop. A rule repository is maintained to store validated heuristics together with lightweight instance-level metadata, facilitating retrieval-based warm starts. The Deliberative Stream improves the active rule asynchronously using compact summaries of recent states and example actions, together with constrained code generation and sandbox evaluation procedures.

4.1 Asynchronous Dual Stream Architecture

We model DFJSP as a Semi-Markov Decision Process with state s_t and action a_t at each decision epoch t [Chang *et al.*, 2022]. The Reactive Stream maps s_t to a dispatching action using a symbolic rule, while the Deliberative Stream performs rule updates using LLM-based reasoning, which operates on a slower timescale.

Reactive Stream for Execution

The Reactive Stream runs inside the scheduling loop, with the objective of confining the decision latency within the millisecond range [Ferreira *et al.*, 2022]. It executes a symbolic priority function f_ϕ implemented as a compiled Python rule derived from the current code representation. At decision epoch t, it selects

$$a_t = \arg\max_{a \in \mathcal{A}_t} f_\phi(s_t, a) \tag{1}$$

The interpreter is constrained to use only observable features in s_t and does not permit side effects, thereby the rule remains deterministic and operates with millisecond-scale execution time.

Deliberative Reasoning Stream for Policy Evolution

The Deliberative Stream updates the symbolic rule off the critical path. It takes a compact summary of recent states and example actions together with the current rule ϕ_{old} and retrieved knowledge $\mathcal{K}$ and proposes a new rule [Yang *et al.*, 2024; Gao *et al.*, 2025]

$$\phi_{\text{new}} \leftarrow \mathcal{P}_{\text{delib}}(\xi, \phi_{\text{old}}, \mathcal{K}) \tag{2}$$

where ξ denotes the summarized observation window. The Reactive Stream keeps running ϕ_{old} while the proposal is generated and tested in the sandbox.

Trigger Update Protocol and Atomic Transition

Deliberation is triggered either every K decision steps or when a monitored metric drops relative to its sliding-window best of length L. Let m_t denote the metric and $m_t^* = \max_{i\in[t-L+1,t]} m_i$. Define the periodic trigger indicator as

$$\mathbb{I}_{\text{periodic}}(t) = \mathbb{I}\{t \bmod K = 0\}, \tag{3}$$

and the performance-based trigger indicator as

$$\mathbb{I}_{\text{perf}}(t) = \mathbb{I}\left\{\frac{m_t^* - m_t}{|m_t^*|} \geq \epsilon\right\}. \tag{4}$$

When $\mathbb{I}_{\text{periodic}}(t) \vee \mathbb{I}_{\text{perf}}(t)$ holds, the Deliberative Stream proposes ϕ_{new} and atomically swaps it in only if sandbox screening accepts it.

4.2 The Agentic Loop: A Safety-Constrained Evolutionary Cycle

The Strategic Planner produces a compact directive based on the current system profile. The Syntactic Enforcer converts this directive into an executable priority dispatching rule, subject to syntactic safety and complexity constraints. The Simulation Validator evaluates candidate rules in a sandbox, allowing an update only if it meets predefined feasibility and improvement criteria. Concurrently, the Semantic Reflector records concise diagnostics to justify the acceptance or rejection decision for human inspection.

Strategic Planner for Bottleneck-Aware Reasoning

The Strategic Planner generates a compact profile by summarizing the current state and formulates a local strategy, such as shifting priority toward the current bottleneck or favoring short remaining processing time on a congested machine. We formalize this mechanism as

$$\mathcal{S}_{plan} \leftarrow \text{Planner}(s_{profile}, \mathcal{M}_{obj}) \tag{5}$$

where $\mathcal{M}_{obj}$ is the objective and $\mathcal{S}_{plan}$ is a short directive consumed by the Syntactic Enforcer.

Syntactic Enforcer for Executable Logic Generation

The Syntactic Enforcer converts the strategic plan $\mathcal{S}_{plan}$ into executable logic in the form of PDRs. It produces symbolic Python functions whose source code can be inspected by domain experts.

Rule generation relies on constrained prompting, which encourages the adoption of a standardized set of shop floor attributes and per operation features. The Coder outputs a priority function f_ϕ that scores each feasible operation and induces a dispatching decision.

The Syntactic Enforcer enforces basic safety and executability by compiling candidate code in a restricted execution context with a limited set of built-in functions and restricted import permissions. If the generated code exceeds a fixed length threshold, it is truncated before compilation, and non-compilable candidates are discarded. The module further extracts lightweight complexity statistics for bookkeeping and subsequent downstream selection.

Simulation Validator for Sandbox Screening

The Simulation Validator screens a candidate policy ϕ_{cand} against the incumbent ϕ_{old} in a sandbox before deployment [Liu *et al.*, 2023]. On an evaluation pool $\mathcal{D}_{val}$, it runs rollouts for both policies under the same decision constraints and disturbance process, yielding per episode objective values $\{y_i^{old}\}_{i=1}^n$ and $\{y_i^{cand}\}_{i=1}^n$. Define the empirical estimate

$$\widehat{\mathcal{M}}_{obj}(\phi) = \frac{1}{n}\sum_{i=1}^{n} y_i^{\phi}. \tag{6}$$

For a minimization objective, the relative improvement is

$$r_t = \frac{\widehat{\mathcal{M}}_{obj}(\phi_{old}) - \widehat{\mathcal{M}}_{obj}(\phi_{cand})}{d_t}, \tag{7}$$

with

$$d_t = \begin{cases} \left|\widehat{\mathcal{M}}_{obj}(\phi_{old})\right|, & \left|\widehat{\mathcal{M}}_{obj}(\phi_{old})\right| > 10^{-9} \\ 1.0, & \text{otherwise.} \end{cases} \tag{8}$$

Let μ_{old}, μ_{cand} and s_{old}^2, s_{cand}^2 be the sample means and variances of the two rollout sets. The validator computes

$$s_p^2 = \frac{(n-1)s_{old}^2 + (n-1)s_{cand}^2}{2n-2}, \tag{9}$$

$$d_t^{eff} = \frac{\mu_{old} - \mu_{cand}}{\sqrt{s_p^2}}, \tag{10}$$

and

$$T_t = \frac{\mu_{old} - \mu_{cand}}{\sqrt{s_{old}^2/n + s_{cand}^2/n}}. \tag{11}$$

A candidate is accepted only if sandbox execution is feasible and

$$\mathbb{I}_{acc} = \mathbb{I}\left\{\begin{array}{l} r_t \geq \epsilon_{rel} \ \wedge \ d_t^{eff} \geq d_{min} \\ \wedge \ T_t \geq T_\alpha \end{array}\right\}, \tag{12}$$

where ϵ_{rel}, d_{min}, and T_α are fixed thresholds. The validator returns $\mathcal{E}_{cand}$ containing $\widehat{\mathcal{M}}_{obj}(\phi_{old})$, $\widehat{\mathcal{M}}_{obj}(\phi_{cand})$, r_t, d_t^{eff}, the number of episodes used, and $\mathbb{I}_{acc}$.

Semantic Reflector for Feedback and Validation

The Semantic Reflector completes the optimization loop by converting sandbox evaluation data into actionable feedback for the next iteration. Functioning as an interpreter, it synthesizes observed tradeoffs and provides a concise diagnostic assessment that rationalizes the acceptance or rejection of each candidate.

This diagnostic process maps sandbox evaluation statistics and high-level policy descriptions into structured feedback. In cases where a candidate policy improves makespan but degrades maximum tardiness, the module interprets the statistical summaries under the chosen objective, produces concise comments that highlight the tradeoff, and determines whether further exploration is warranted.

Diagnostic feedback is logged in the trajectory records, where it serves to rationalize the acceptance or rejection of individual candidates. Its decision signal controls whether the search runs another iteration. The final rule selection, meanwhile, depends solely on sandbox evaluation results and a complexity term.

4.3 Rule Repository for Lifelong Learning

To mitigate knowledge volatility inherent in conventional reinforcement learning and enhance adaptation efficiency across heterogeneous environments, we introduce the Semantic Knowledge Repository. This module persists and retrieves optimized symbolic heuristics, serving as reusable knowledge for RACE-Sched. Through the decoupling of knowledge retention from parameter updates, it preserves high-performing heuristics under nonstationary dynamics. During runtime, the retrieval of validated priors enables faster adaptation than reoptimizing from scratch.

Formal Structure of the Knowledge Repository

The Semantic Knowledge Repository is organized as an indexed library of interpretable symbolic heuristics. Each entry constitutes a tuple $\mathcal{T}_i = \langle \mathbf{m}_i, \phi_i, v_i, \mathcal{C}_i \rangle$ that links a synthesized policy to its operational context. The meta feature vector $\mathbf{m}_i$ encodes global problem topology including job and machine cardinality to facilitate similarity based retrieval within the vector space. The component ϕ_i represents the executable policy source code while the scalar v_i quantifies the normalized performance improvement over the baseline. To promote generalizability we incorporate a structural complexity score $\mathcal{C}_i$ derived from code statistics such as operator count and branching factor. This metric functions as a regularization penalty during retrieval and prioritizes parsimonious logic that avoids overfitting to specific training instances.

Retrieval and Cross Scenario Transfer

Upon encountering a new problem instance $\mathbf{m}_{new}$, the system initiates a warm start by retrieving the nearest policy neighbors from the repository. The retrieval score $S(\mathcal{T}_i|\mathbf{m}_{new})$ balances historical efficacy, feature alignment, and complexity:

$$S(\mathcal{T}_i|\mathbf{m}_{new}) = v_i - \lambda_1 D(\mathbf{m}_{new}, \mathbf{m}_i) - \lambda_2 \log(1+\mathcal{C}_i) + \delta_{match} \tag{13}$$

where v_i is the policy efficacy, $D(\mathbf{m}_{new}, \mathbf{m}_i)$ is the feature distance between the new instance $\mathbf{m}_{new}$ and repository entry $\mathbf{m}_i$, and $\mathcal{C}_i$ is the policy complexity.

The distance $D(\mathbf{m}_{new}, \mathbf{m}_i)$ is calculated as:

$$D(\mathbf{m}_{new}, \mathbf{m}_i) = \frac{|N_j^{(new)} - N_j^{(i)}|}{N_j^{(new)}} + \frac{|N_m^{(new)} - N_m^{(i)}|}{N_m^{(new)}} \tag{14}$$

where N_j and N_m represent job volume and machine capacity, respectively.

This regularization prevents maladaptive transfer by penalizing shifts in job volume and machine capacity.

5 Experiments

5.1 Experimental Setup

Benchmark Datasets and Evaluation Metrics

We validate performance across three benchmark suites encompassing standardized testbeds to high-fidelity simulation settings. GEN-Bench assesses policy generalization capabilities, while MK-Bench extends classical Brandimarte instances to the dynamic domain [Brandimarte, 1993]. JMS-Bench introduces industrial complexity through the simulation of semiconductor cluster tools with reentrant flows [Cao and Yuan, 2025]. Regarding evaluation metrics, we measure solution quality using the Relative Percent Deviation (RPD) from the Best Known Solution and the average rank across all test instances. We further perform an analysis of cumulative token consumption and wall-clock time to assess the tradeoff between computational efficiency and real-time responsiveness.

Baseline Methods and LLM Backends

We evaluate RACE-Sched against a spectrum of symbolic, neural, and generative baselines. Classical approaches are exemplified by a GP dispatcher [Mei *et al.*, 2016]. Neural methods encompass the attention-based DAN [Wang *et al.*, 2024b] alongside three Reinforcement Learning agents: the breakdown robust IDDQN [Wu *et al.*, 2025], the option-based PPO-OC [Yuan *et al.*, 2025], and the hierarchical HMP-SAC [Ding *et al.*, 2025]. Finally, we compare against two LLM baselines: a zero-shot LLM-Direct agent and Reflec-Sched, which employs reflection-driven simulations [Cao and Yuan, 2025].

The proprietary GPT models include GPT-4o [Hurst *et al.*, 2024] and GPT-5 Nano [Singh *et al.*, 2025], while the open-weight models cover DeepSeek-V3 [DeepSeek-AI, 2024], DeepSeek-V3.2 [DeepSeek-AI, 2025], and the Qwen3 series with 8B, 14B, 32B parameters and Qwen3-Coder-30B [Yang *et al.*, 2025]. Inference is managed by using the vLLM [Kwon *et al.*, 2023] high-throughput serving engine, hosted on a compute node configured with an Intel Xeon Platinum 8468V CPU and an NVIDIA H100 80GB GPU.

5.2 Performance and Efficiency

Scheduling Performance Analysis

As shown in Table 1, in normal scale scenarios, where combinatorial complexity is most pronounced, RACE-Sched exhibits superior convergence performance toward global opti-

Category	Model Backend	GEN-Bench Small		GEN-Bench Normal		MK-Bench		JMS-Bench	
		RPD (%)	Rank	RPD (%)	Rank	RPD (%)	Rank	RPD (%)	Rank
Baselines	GP	39.20	20.50	72.41	23.00	62.88	17.20	51.25	18.30
	HMPSAC	12.03	12.06	12.34	14.55	20.17	11.80	12.87	9.20
	IDDQN	13.58	14.11	9.82	12.10	17.47	11.20	11.02	8.30
	DAN	13.47	13.44	10.40	12.55	23.74	14.00	18.88	13.60
	PPO-OC	17.33	15.72	12.23	15.25	21.60	11.60	14.13	10.60
LLM-Direct	Qwen3-8B	11.13	14.50	10.15	13.00	29.62	15.30	15.77	12.40
	Qwen3-14B	7.86	8.39	10.48	13.55	13.44	7.80	16.57	12.20
	Qwen3-32B	9.47	11.17	10.70	13.30	23.58	13.40	11.98	8.60
ReflecSched	Qwen3-8B	8.73	10.56	7.44	9.60	15.94	9.10	9.56	7.70
	Qwen3-14B	6.71	7.11	7.67	9.55	20.20	11.60	9.29	7.50
	Qwen3-32B	7.10	8.50	8.82	11.55	19.94	10.70	10.89	8.20
	DeepSeek-V3.2	9.56	9.56	8.24	11.35	12.33	8.10	11.71	8.00
	GPT-5 Nano	9.01	9.61	9.38	11.00	18.89	11.30	13.41	9.50
RACE-Sched (Ours)	Qwen3-8B	9.54	11.39	5.33	6.75	6.43	4.80	11.09	9.50
	Qwen3-14B	10.71	11.83	6.31	7.50	7.46	4.90	9.37	6.10
	Qwen3-32B	9.72	11.50	6.89	8.15	7.31	4.50	10.67	7.30
	DeepSeek-V3.2	8.02	9.22	7.35	9.30	7.87	5.40	11.01	7.90
	GPT-5 Nano	8.25	9.39	6.44	7.55	10.75	6.60	9.73	6.10
	Qwen3-Coder-30B	4.97	6.50	8.89	11.65	11.23	7.80	13.42	10.50

Table 1: Performance on GEN-Bench, MK-Bench, and JMS-Bench. Lower RPD and rank are better.

mality. Specifically, the Qwen3-8b variant achieves an average RPD of 5.33%, outperforming the leading DRL baseline IDDQN with RPD 9.82% by 45.7%. This performance difference highlights the capacity of symbolic evolution to address long-horizon strategic dependencies, which are often obscured in the latent representations of neural approximations.

In Normal scale instances, RACE-Sched variants consistently occupy the upper tiers of the performance hierarchy. Specifically, the Qwen3-8b configuration attains a mean rank of 6.75, maintaining a significant lead over the top performing DRL baselines, IDDQN with rank 12.1 and DAN with rank 12.55. In Small scale instances, despite the compressed optimization margins characteristic of smaller search spaces, RACE-Sched retains a distinct performance over LLM-Direct and ReflecSched. This cross scale stability underscores that the synthesized symbolic code captures fundamental scheduling principles, enabling generalization that remains invariant to shifts in job volume and machine density.

Token Consumption Assessment

Table 2 presents a comparative evaluation of cumulative token expenditure. The Semantic Knowledge Repository significantly reduces reliance on token-intensive generative exploration. Across all model backends ranging from Qwen3 to GPT-5, RACE-Sched consistently requires the lowest input and output volume. Specifically, under the Qwen3-8b configuration, our framework lowers input token consumption by approximately 60% relative to LLM-Direct baseline. This efficiency gain arises from two structural advantages. First, the retrieval mechanism utilizes archived priors to prune the search space and eliminate redundant inferential steps. Second, the transition from per step action generation to compact policy synthesis drastically minimizes output volume. By confining LLM activation to performance critical triggers, the architecture effectively balances the complexity associated with deliberative reasoning and computational sustainability.

Real-Time Latency Evaluation

Table 3 highlights the significant discrepancy between heuristic execution and deliberative reasoning. The Reactive Stream consistently exhibits sub-millisecond latency across all benchmark suites. Notably, the DeepSeek V3.2 and GPT-5 Nano configurations achieve a mean response time of less than 0.1 ms. This deterministic performance ensures that the tactical control loop complies with the strict real-time constraints imposed by industrial manufacturing environments.

In contrast, the Deliberative Stream functions on a timescale that is orders of magnitude longer with latencies ranging from 77 to 114 seconds per policy update. Synchronous integration of such high-latency reasoning would result in catastrophic blockages of the production line. RACE-Sched mitigates this risk by isolating the Deliberative Stream from the critical execution path. New rules are deployed only after validation via an atomic update. This architectural design maintains dispatching latency at the microsecond level, while enabling continuous long-horizon optimization.

5.3 Ablation Studies

We compare four structural variants.

- **Full**: The complete architecture integrating both repository retrieval and iterative deliberation for continuous

Model Backend	Mean Input Tokens			Mean Output Tokens		
	LLM-Direct	ReflecSched	RACE-Sched	LLM-Direct	ReflecSched	RACE-Sched
Qwen3-8B	59,977	69,386	24,609	12,261	11,679	4,802
Qwen3-14B	62,910	67,091	47,581	9,423	18,725	7,416
Qwen3-32B	78,832	68,905	27,714	14,653	32,309	3,900
DeepSeek-V3 / V3.2	57,625	69,245	44,565	14,277	8,488	8,464
GPT-4o	58,009	64,031	35,494	10,544	18,307	4,675
GPT-5 Nano	N/A	70,435	28,683	N/A	108,403	50,353

Table 2: Model-wise comparison of token consumption efficiency on GEN-Bench.

Backend	Datasets	React ms	Delib s	Ratio $\times 10^6$
DeepSeek-V3.2	GEN-Bench	0.054	77.05	1.43
	MK-Bench	0.084	113.79	1.35
	JMS-Bench	0.092	90.08	0.98
GPT-5 Nano	GEN-Bench	0.050	114.08	2.28
	MK-Bench	0.060	111.94	1.87
	JMS-Bench	0.085	109.39	1.29

Table 3: Reactive Stream latency and Deliberative update time across model backends.

rule optimization.

- **w/o Loop**: Disables the reasoning loop to generate a single candidate rule per step without retrospective feedback.
- **w/o Repo**: Removes knowledge retrieval to force a cold start initialization without historical priors.
- **w/o Both**: Excludes both components to serve as a baseline lacking memory and self correction mechanisms.

We retrieve archived rules based on normalized Manhattan distance within a meta-feature space defined by job and machine counts. To ensure reproducibility during benchmarking, a serialized execution protocol is implemented, while the deployed system maintains asynchronous decoupling.

Contribution of Epistemic Transfer via Rule Repository
Table 4 highlights the critical role of the Semantic Knowledge Repository in mitigating cold start inefficiencies. The comparison between the Full architecture and the variant excluding the repository reveals a substantial performance gap. On the GEN-Bench Small dataset, the retrieval of historical priors reduces the Relative Percent Deviation from 26.19% to 10.41%. This finding indicates that initializing deliberative reasoning with heuristics derived from similar problem instances effectively prunes the search space and minimizes reliance on stochastic exploration. Although the magnitude of improvement varies across specific datasets, the repository provides significant stability on a global scale. The Consolidated metrics confirm this structural advantage: the Full model achieves a global average deviation of 9.39%, in contrast to 14.07% for the unaugmented baseline.

Impact of Recursive Agentic Refinement
A comparison between the Full model and the variant without the loop reveals that refinement generally improves solution quality for complex tasks. For instance, on the GEN-Bench Normal dataset, enabling the loop reduces the RPD from 8.17% to 6.16%. This gain stems from the capability of the system to analyze sandbox feedback and rectify logical inconsistencies in subsequent iterations. Although the single-step approach performs competitively on simpler tasks, the Full architecture secures a superior consolidated average rank of 2.32 compared to 2.40 for the variant lacking the loop. This confirms that iterative validation provides greater stability across diverse problem distributions.

Dataset	Metric	Full	No Repo	No Loop	No Both
GEN-S	RPD (%)	10.41	26.19	10.18	31.02
	Rank	1.94	2.61	1.78	3.00
GEN-N	RPD (%)	6.16	9.54	8.17	13.77
	Rank	2.15	3.25	2.50	3.60
MK	RPD (%)	11.78	9.43	14.16	16.21
	Rank	2.20	2.40	2.60	3.20
JMS	RPD (%)	9.22	11.10	8.52	14.80
	Rank	3.00	2.70	2.70	2.90
Global Consol.	Avg. RPD (%)	9.39	14.07	10.26	18.95
	Avg. Rank	2.32	2.74	2.40	3.18

Table 4: Ablation results quantifying the contribution of the Repository and Iterative Loop. GEN-S and GEN-N denote the Small and Normal subsets of GEN-Bench, respectively.

5.4 Resilience Under Machine Failures

We evaluate operational robustness under nonstationary conditions using a stress test within the JMS Bench framework. Figure 2 depicts the system's response to an abrupt capacity loss at a critical workstation, which occurs at time $T_{fail} = 12.0$.

The baseline approaches demonstrate inadequate adaptability to this structural shift. The Single Cold variant results in the worst makespan of 35.0 due to its reliance on a static heuristic that cannot adjust to the new bottleneck. Correspondingly, the DRL baseline HMPSAC exhibits high variance and a prolonged recovery period, which suggests insufficient generalization to out-of-distribution states.

In contrast, RACE-Sched demonstrates superior resilience. Upon detection of the throughput degradation, the Deliberative Stream asynchronously synthesizes and deploys a new

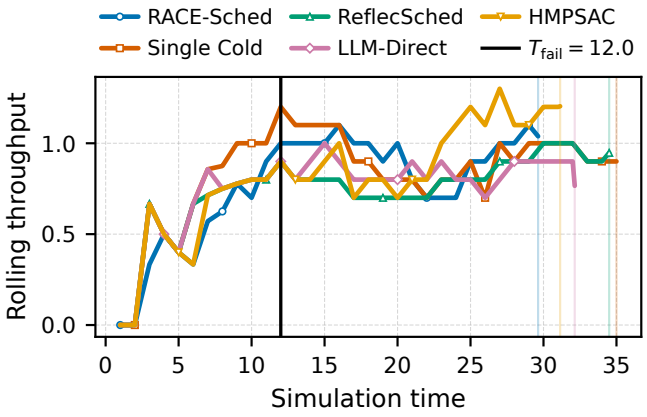

Figure 2: Rolling throughput after a machine failure.

priority rule tailored to the altered system dynamics. This rapid adaptive response restores throughput to pre-failure levels and results in the minimum final makespan of 29.6. These results confirm that the asynchronous decoupling effectively supports continuous policy refinement without blocking the real-time control loop.

6 Conclusion

We introduce RACE-Sched to reconcile the reasoning capability of LLMs with the latency constraints of industrial control systems. By decoupling fast heuristic execution from asynchronous policy synthesis, RACE-Sched maintains millisecond-level responsiveness while continuously evolving symbolic control logic. The Semantic Knowledge Repository reduces redundant deliberation through retrieval-based reuse, and sandbox validation prevents unverified rules from entering the online control loop. Extensive experiments across GEN-Bench, MK-Bench, and JMS-Bench show that RACE-Sched outperforms state-of-the-art DRL and generative scheduling baselines while preserving low-latency online execution.

RACE-Sched also has several deployment limitations. Its rule evolution quality depends on the capability of the background LLM, since weaker models may generate invalid or sub-optimal candidate rules that are rejected by sandbox validation. The construction of representative sandbox scenarios and trigger conditions requires engineering effort, because the validation environment must reflect the main disturbance patterns of the target shop floor. Although the asynchronous design prevents deliberation from blocking the control loop, reaction to brand-new disturbance types can still be delayed when candidate synthesis and validation require multiple deliberation cycles.

Ethical Statement

There are no ethical issues.

Acknowledgments

This work was partially supported by Shenzhen Loop Area Institute under Project No. FP202602. The authors gratefully acknowledge the support from the project team members involved in this work.

References

[Brandimarte, 1993] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.*, 41(3):157–183, 1993.

[Cao and Yuan, 2025] Shijie Cao and Yuan Yuan. Reflecsched: Solving dynamic flexible job-shop scheduling via LLM-powered hierarchical reflection. *CoRR*, abs/2508.01724, 2025.

[Cao *et al.*, 2023] Shijie Cao, Rui Li, Wenyin Gong, and Chao Lu. Inverse model and adaptive neighborhood search based cooperative optimizer for energy-efficient distributed flexible job shop scheduling. *Swarm Evol. Comput.*, 83:101419, 2023.

[Chang *et al.*, 2022] Jingru Chang, Dong Yu, Yi Hu, Wuwei He, and Haoyu Yu. Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. *Processes*, 10(4):760, 2022.

[Chen *et al.*, 2026] Hao Chen, Jiaming Liu, Chenyang Gu, Zhuoyang Liu, Renrui Zhang, Xiaoqi Li, Xiao He, Yandong Guo, Chi-Wing Fu, Shanghang Zhang, and Pheng-Ann Heng. Fast-in-Slow: A dual-system VLA model unifying fast manipulation within slow reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[DeepSeek-AI, 2024] DeepSeek-AI. DeepSeek-V3 technical report. *CoRR*, abs/2412.19437, 2024.

[DeepSeek-AI, 2025] DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *CoRR*, abs/2512.02556, 2025.

[Ding *et al.*, 2025] Linshan Ding, Zailin Guan, Dan Luo, and Lei Yue. Data-driven hierarchical multi-policy deep reinforcement learning framework for multi-objective multiplicity dynamic flexible job shop scheduling. *Journal of Manufacturing Systems*, 80:536–562, 2025.

[Ferreira *et al.*, 2022] Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. *Omega*, 111:102643, 2022.

[Gao *et al.*, 2025] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: On path to artificial super intelligence. *CoRR*, abs/2507.21046, 2025.

[Holthaus and Rajendran, 2000] Oliver Holthaus and Chandrasekharan Rajendran. Efficient jobshop dispatching rules: Further developments. *Production Planning & Control*, 11(2):171–178, 2000.

[Hurst *et al.*, 2024] Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *CoRR*, abs/2410.21276, 2024.

[Karami *et al.*, 2025] Rachid Karami, Rajeev Patwari, Hyoukjun Kwon, and Ashish Sirasao. Exploring the dynamic scheduling space of real-time generative AI applications on emerging heterogeneous systems. *CoRR*, abs/2507.14715, 2025.

[Kwon *et al.*, 2023] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace, editors, *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM, 2023.

[Li *et al.*, 2026] Kai Li, Bao Zheng, Liping Xu, Fulong Xie, and Zhicheng Wang. Multi-objective dynamic flexible job shop scheduling using multi-head network-based deep reinforcement learning. *Expert Systems with Applications*, 298:129542, 2026.

[Liang *et al.*, 2023] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 9493–9500. IEEE, 2023.

[Liu *et al.*, 2023] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, volume 36, pages 21558–21572, 2023.

[Mei *et al.*, 2016] Yi Mei, Mengjie Zhang, and Su Nguyen. Feature selection in evolving job shop dispatching rules with genetic programming. In Tobias Friedrich, Frank Neumann, and Andrew M. Sutton, editors, *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 365–372. ACM, 2016.

[Ouelhadj and Petrovic, 2009] Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *J. of Scheduling*, 12(4):417–431, August 2009.

[Priore *et al.*, 2014] Paolo Priore, Alberto Gómez, Raúl Pino, and Rafael Rosillo. Dynamic scheduling of manufacturing systems using machine learning: An updated review. *Ai Edam*, 28(1):83–97, 2014.

[Shah *et al.*, 2026] Ankit Parag Shah, Mohammad-Parsa Hosseini, Su Min Park, Connie Miao, and Wei Wei. Small language models: Architecture, evolution, and the future of artificial intelligence. *Preprints*, January 2026.

[Singh *et al.*, 2025] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. OpenAI GPT-5 System Card. *arXiv preprint arXiv:2601.03267*, 2025.

[Singh *et al.*, 2026] Punit Singh, Krishna Krishnan, and Enkhsaikhan Boldsaikhan. A review of production scheduling with artificial intelligence and digital twins. *Journal of Manufacturing and Materials Processing*, 10(1):6, 2026.

[Wang *et al.*, 2024a] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

[Wang *et al.*, 2024b] Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. Flexible job shop scheduling via dual attention network-based reinforcement learning. *IEEE Trans. Neural Networks Learn. Syst.*, 35(3):3091–3102, 2024.

[Wei *et al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, volume 35, pages 24824–24837, 2022.

[Wu *et al.*, 2025] Rui Wu, Jianxin Zheng, Xixing Li, Hongtao Tang, Xi Vincent Wang, and Yibing Li. Dynamic scheduling for flexible job shop under machine breakdown using improved double deep q-network. *Expert Syst. Appl.*, 288:128280, 2025.

[Xu *et al.*, 2025] Meng Xu, Yi Mei, Fangfang Zhang, and Mengjie Zhang. Learn to optimise for job shop scheduling: a survey with comparison between genetic programming and reinforcement learning. *Artif. Intell. Rev.*, 58(6):160, 2025.

[Yang *et al.*, 2024] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.

[Yang *et al.*, 2025] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[Yao *et al.*, 2023] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and

Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, volume 36, pages 11809–11822, 2023.

[Yuan *et al.*, 2025] Minghai Yuan, Qi Yu, Lizhi Zhang, Songwei Lu, Zichen Li, and Fengque Pei. Deep reinforcement learning based proximal policy optimization algorithm for dynamic job shop scheduling. *Comput. Oper. Res.*, 183:107149, 2025.

[Zhang *et al.*, 2020] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, volume 33, pages 1621–1632, 2020.

[Zhang *et al.*, 2023] Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. Multitask multiobjective genetic programming for automated scheduling heuristic learning in dynamic flexible job-shop scheduling. *IEEE Trans. Cybern.*, 53(7):4473–4486, 2023.

[Zhang *et al.*, 2025] Yuzhi Zhang, Shidu Dong, Zhenfang Yuan, Ting Wen, Jianfeng Xiao, and Zhuo Diao. Meta-relation-based heterogeneous graph neural network with deep reinforcement learning for flexible job shop scheduling. *Expert Systems with Applications*, 291:128411, 2025.