ALIGN-AND-IMPROVE: ADDRESSING STATE STALENESS AND PLAN QUALITY IN DUAL-STREAM DYNAMIC SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic job shop scheduling requires immediate assignments, while slower search constructs candidate rules and plans. In a dual-stream scheduler, a request may be based on an observation that becomes stale before generation starts. A returned rule may also produce an inferior continuation, and an accepted plan may lose its advantage as execution proceeds. We present **A**lign-**a**nd-**I**mprove (AAI), which addresses these distinct sources of mismatch at request, plan and action time. Late State Binding (LSB) associates a pending request with the latest observed state when the generator claims it. Verified Plan Update (VPU) materializes candidates on the current state and replaces an incumbent only after a feasible, strictly improving comparison and a commitment-time consistency check. Rollout-Based Action Arbitration (RAA) compares the retained plan's next action with an action from the fast dispatching rule, evaluating both with the same fast-policy continuation before executing one assignment. Across 58 instances from four benchmark families, AAI lowers mean relative percentage deviation by about five percentage points versus RACE on three shared Qwen3.5 backbones while cutting model-token usage by up to about 90%.

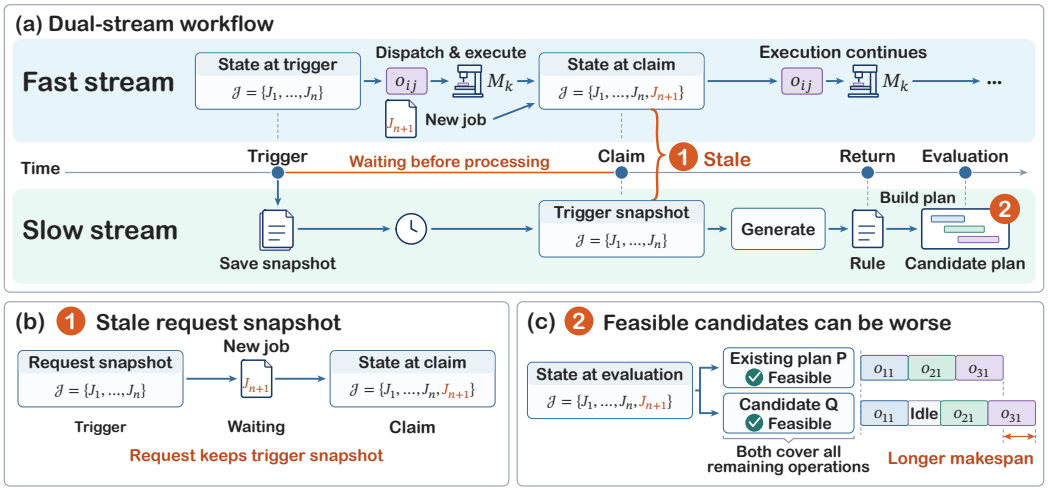


Figure 1: Two limitations of asynchronous dual-stream scheduling. (a) Fast-stream dispatches and new job arrivals can change the state while a slow-stream request waits or is processed. (b) A request that retains its trigger-time snapshot may therefore omit jobs visible when processing begins. (c) Feasibility alone does not establish plan quality: a candidate covering all remaining operations can still have a longer makespan than the existing plan.

1 INTRODUCTION

Dynamic job shop scheduling requires making operation assignments promptly, while seeking to minimize makespan (Xu et al., 2025). In a dual-stream scheduler, the fast stream executes dispatching rules for immediate assignments, while the slow stream invokes a large language model (LLM)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

to generate candidate rules and constructs candidate plans asynchronously (Cao et al., 2026). The fast stream keeps machines active while slower search explores alternatives beyond the immediate dispatching decision. Returned rules are evaluated on the current state and can supply candidate plans for subsequent decisions. However, fast-stream decisions keep executing while a slow-stream request is pending or is being processed, causing the scheduling state to evolve before the proposal returns. Accordingly, the core research question is not merely how to generate better candidates, but to determine when a slow-stream request remains relevant and whether its output deserves deployment.

State relevance and plan quality require distinct forms of control. By the time LLM generation begins, a request may carry an obsolete state reference, whereas even a feasible candidate with respect to the current state may perform worse than the continuation already in place. Our asynchronous diagnostic reveals that stale returns arise even in the absence of queue drops, while state-local replays yield candidates that both improve and worsen the baseline continuation. The first observation motivates delaying state binding until the request is claimed. The second motivates comparing complete continuations before allowing a candidate to supersede the current plan. Yet acceptance is a state-local decision: subsequent dispatches or new observations may change the benefit of the retained plan's next assignment. This motivates a further action-level comparison during execution. A more up-to-date state reference does not guarantee a better candidate, and filtering out poor candidates cannot compensate for stale information used during candidate construction.

Align-and-Improve (AAI) implements this separation through Late State Binding (LSB) and Verified Plan Update (VPU). LSB retains an LLM request intent without a snapshot, then binds the request to the coordinator's latest observed snapshot upon successful claim. VPU materializes returned rules and native-search candidates on the current state and replaces the incumbent plan only when a supported comparison establishes feasibility and a strict reduction in the remaining makespan. Prior to commitment, AAI rechecks the relevant state and plan identities. At execution time, Rollout-Based Action Arbitration (RAA) compares the candidate action with an action from the fast dispatching rule, using that same rule for subsequent rollout decisions. Its action-level check addresses the possibility that a previously accepted plan no longer offers a useful next assignment.

Our contributions are as follows:

- We identify two distinct limitations in dual-stream scheduling. Request state references can become stale before generation begins, while feasible candidates can still underperform relative to the existing continuation.
- We introduce AAI to address these limitations through two complementary mechanisms. LSB delays state binding until a request is claimed, while VPU permits candidate replacement only after establishing feasibility and a strict reduction in the remaining makespan.
- We evaluate AAI over 58 instances from four benchmark families. AAI reduces mean RPD and model-token usage relative to RACE on the three shared backbones. Component and controlled timing studies examine VPU and LSB, while a comparison with PDR-only dispatching rules probes whether a classical fast rule alone accounts for the observed scheduling quality.

2 Related Work

Learning-based methods and program search offer different ways to construct scheduling policies. L2D learns dispatching policies from graph representations (Zhang et al., 2020), while later schedulers use topological graph representations (Zhang et al., 2024) or learn multiple dispatch policies (Choi et al., 2025; Grinsztajn et al., 2023). Program-search methods instead refine executable rules: FunSearch searches program space (Romera-Paredes et al., 2024), EoH evolves heuristic descriptions and implementations (Liu et al., 2024), and ReEvo uses reflective feedback (Ye et al., 2024). LLMs have also been used to evolve heuristic optimizers (Shi et al., 2026) and guide neighborhood search for scheduling (Zhang et al., 2025).

Using a candidate during execution raises a separate question. Rolling-horizon optimization revises plans over successive subproblems (Li et al., 2025), and RACE combines fast dispatching with asynchronous LLM-generated proposals (Cao et al., 2026). In this setting, a proposal may be based

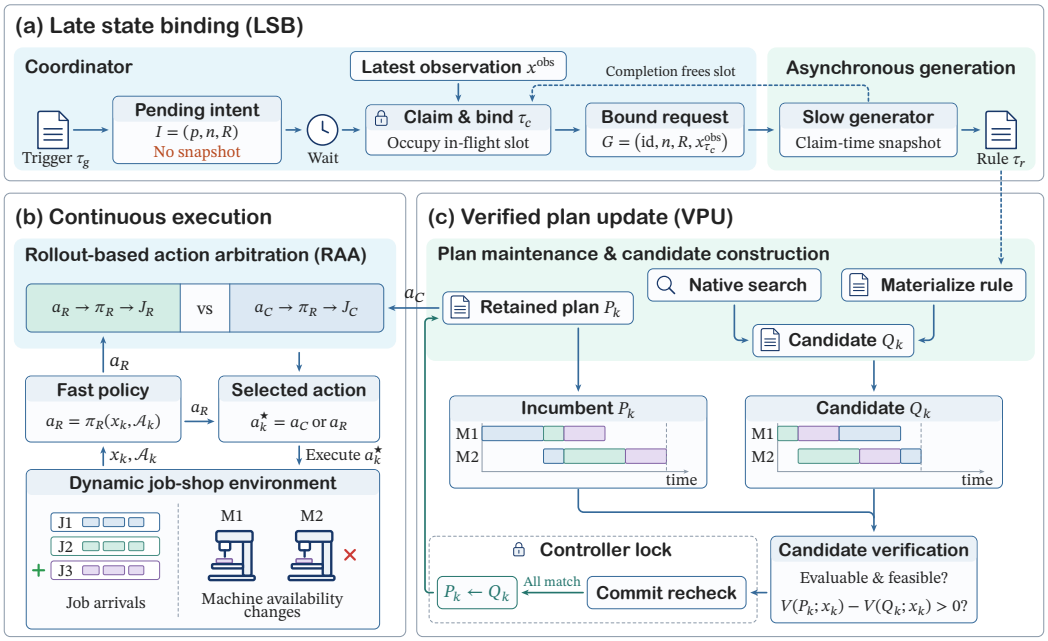


Figure 2: AAI workflow. (a) LSB records a pending request without a snapshot and binds it to the latest observed state when claimed. (b) During execution, RAA compares the retained plan's next action with a fast-rule action under the same continuation policy before dispatching. (c) VPU evaluates feasible candidates against the incumbent and replaces the retained plan only after a strict improvement and a commitment-time consistency check.

on an outdated state, and a feasible plan may still be worse than the incumbent continuation. AAI addresses these two conditions when binding requests and updating plans, then checks the retained plan's next action during execution.

3 Problem Formulation

Consider a dynamic job shop with a set of jobs $\mathcal{J}$ and a set of machines $\mathcal{M}$. Each job $j \in \mathcal{J}$ has a release time r_j and consists of an ordered sequence of operations $\mathcal{O}_j$. Each operation o must be assigned to an eligible machine $m \in \mathcal{M}_o$, with processing time $p_{o,m}$. A feasible schedule satisfies the constraints of job release time, operation precedence, and machine availability. Specifically, an operation can start only after its job has been released and its preceding operation has completed, and each machine can process at most one operation at a time. During execution, dynamic events such as new job arrivals and changes in machine availability can alter the observable scheduling state (Zhou et al., 2026). The objective is to minimize the terminal makespan (Cao et al., 2023)

$$C_{\max} = \max_{j \in \mathcal{J}} C_j, \tag{1}$$

where C_j denotes the completion time of job j.

At decision step k, the scheduler observes the current scheduling state x_k and the set of legal assignments $\mathcal{A}_k$. The state records the current simulation time t_k, observable job and machine information, and the executed prefix. A continuation must remain compatible with this prefix and preserve machine assignments and timing for operations already in progress. A scheduling policy π selects an assignment

$$a_k = \pi(x_k, \mathcal{A}_k) \in \mathcal{A}_k. \tag{2}$$

As execution proceeds and new information becomes observable, both x_k and $\mathcal{A}_k$ may change. Each decision only uses information available at that decision point, with no access to unobserved future events.

In the dual-stream setting considered in this work, the fast stream employs a dispatching policy π_R, instantiated as a classical priority dispatching rule (PDR), to produce an immediate legal assignment

$$a_R = \pi_R(x_k, \mathcal{A}_k) \in \mathcal{A}_k, \tag{3}$$

while the slow stream asynchronously generates a candidate for subsequent scheduling decisions. A slow request is triggered at τ_g, claimed at τ_c, and returns at τ_r, where

$$\tau_g \leq \tau_c \leq \tau_r. \tag{4}$$

The request timings τ_g, τ_c, and τ_r describe the asynchronous request lifecycle, whereas t_k denotes the simulation time used to evaluate scheduling costs. Fast-stream assignments keep executing while the request is pending or being processed, so the scheduling state may evolve before the slow-stream result becomes available.

4 ALIGN-AND-IMPROVE

AAI addresses two limitations in deploying slow-stream proposals: pending requests may reference outdated observations, and the quality of a rule's decisions may vary as execution proceeds, even after a candidate has been accepted. These requirements arise at different stages of the asynchronous workflow. The scheduling state can evolve while a request waits for processing and again before the resulting candidate is evaluated. It can continue to evolve after a rule or plan has been accepted. Moreover, a candidate constructed from recent information is not guaranteed to yield a better schedule.

AAI fulfills these requirements through LSB and VPU. LSB governs the request lifecycle. It records pending work without a snapshot and binds the request to the coordinator's latest observation upon request claim. VPU governs plan replacement. It materializes returned rules on the current state, considers candidates from native search, and compares feasible candidates against a maintained incumbent plan. A candidate replaces the incumbent only if it strictly reduces remaining makespan and the state and plan used for evaluation remain unchanged at commitment.

The retained plan subsequently supplies a candidate action at each decision step. When Rollout-Based Action Arbitration (RAA) is enabled, this action is compared with the fast-stream action through rollouts that share the same continuation policy. The controller then executes the selected action and synchronizes the retained plan with the updated state. Figure 2 summarizes this workflow. VPU controls plan replacement, while RAA controls the first action executed from the retained plan at each decision.

4.1 LATE STATE BINDING

Separating request creation from state binding. A slow-stream request is triggered at time τ_g, yet it may not be claimed for processing until τ_c. Fast-stream assignments and incoming observations can change the scheduling state during this waiting interval. If the request stores a snapshot at τ_g, it retains that outdated reference even when a more recent observation is available at τ_c. LSB therefore separates the decision to request slow-stream work from the decision about which state the request should reference.

Prior to claim, the coordinator stores only a pending intent $I = (p, n, R)$, where $p \in \{0, 1\}$ indicates whether work is pending, n denotes a monotonically increasing sequence number, and R represents a bounded set of trigger reasons. Each trigger sets p to one, increments n, and merges its trigger reason into R without attaching a snapshot. The intent records that work is requested, while the coordinator continues to update its latest observed snapshot x^{obs}.

Claiming and binding a request. Let F denote the in-flight request and B the coordinator's logical budget state. A pending intent can be claimed only when no request is already in flight, an observation is available, and the request budget permits additional work:

$$(F = \varnothing) \wedge (p = 1) \wedge (x^{\mathrm{obs}} \neq \varnothing) \wedge \mathrm{budgetOK}(B). \tag{5}$$

Here, $\mathrm{budgetOK}(B)$ denotes the eligibility check under the configured request budget. When these conditions hold, the coordinator atomically consumes the pending intent, binds the latest observed snapshot, and occupies the in-flight slot with

$$G = (\text{request id}, n, R, x^{\mathrm{obs}}_{\tau_c}). \tag{6}$$

The eligibility check and these updates occur under the same coordinator lock, preventing a coordinator update from intervening between the check and state binding.

The coordinator maintains at most one pending intent and one logical in-flight request. Triggers received while a request is in flight are merged into the pending intent instead of creating additional in-flight requests. Once the current request completes, the accumulated intent can be claimed if Equation 5 holds again. That subsequent request is bound to the observation available at its own claim time.

Scope of state alignment. LSB changes the state reference selected at claim, which does not refresh a request continuously during generation. Its benefit therefore depends on whether relevant information becomes observable between τ_g and τ_c. Information arriving after claim is not incorporated through this binding operation.

4.2 Verified Plan Update

LSB determines which observation a request references, but it does not guarantee the quality of the resulting candidate. A candidate may be feasible on the current state and still produce a worse continuation than the available plan. VPU therefore evaluates candidate plans against a maintained incumbent on a common current-state snapshot, and separates plan evaluation from the final commitment of a replacement.

Maintaining a current-state reference plan. At decision step k, let P_k denote the incumbent plan and $\mathcal{O}_k$ the observable unfinished operations, including operations already in progress. The incumbent represents a complete continuation for this scope. Before comparing candidates, VPU synchronizes the incumbent with x_k by removing completed operations, incorporating newly observed work, and preserving the assignments and timing for operations already in progress. When the projected incumbent is infeasible, it is repaired if the public-state model supports a feasible baseline continuation.

Initialization and incumbent maintenance establish the reference against which candidates are compared. They are distinct from candidate-driven replacement and are not subject to its strict-improvement condition. A comparison proceeds only when the incumbent is feasible and the available public-state information is sufficient to evaluate both plans. If either prerequisite fails, VPU skips a candidate-driven replacement.

Constructing and checking candidate plans. VPU obtains candidate plans Q_k by materializing returned rules on the current state and by performing native search. In the online native-search path considered here, it first attempts CP-SAT search subject to current scheduling boundary constraints, with a deterministic neighborhood search as the fallback. Regardless of their origin, candidates are evaluated against the same incumbent and current-state snapshot.

Each plan specifies machine assignments and processing orders over $\mathcal{O}_k$. Start times for unstarted operations follow from these choices and current commitments. Before cost comparison, VPU checks exact coverage of $\mathcal{O}_k$, machine eligibility, finite and consistent processing times, and preservation of running commitments and the executed prefix. The combined job-precedence and machine-order graph must be acyclic; these checks establish whether the candidate admits a feasible continuation under Section 3.

Evaluating both plans on the same state. VPU applies the same temporal evaluation procedure to the incumbent plan and candidate. For a plan P, let $m_P(o)$ denote the machine assigned to operation o, $j(o)$ its job, and $A_m(x_k)$ be the earliest time at which machine m is available for additional processing under the current commitments. Let $\mathrm{Pred}_P(o)$ contain the immediate predecessors of o within $\mathcal{O}_k$, induced by job precedence and the machine order specified by P.

For each operation that has not started, VPU computes its start time s_o and completion time f_o in topological order:

$$\begin{aligned} s_o &= \max\Big\{t_k,\ r_{j(o)},\ A_{m_P(o)}(x_k),\ \max_{q\in\mathrm{Pred}_P(o)} f_q\Big\}, \\ f_o &= s_o + p_{o,m_P(o)}. \end{aligned} \tag{7}$$

The terms require the operation to start no earlier than the current simulation time, the release time of its associated job, the ready time of its assigned machine, and the completion time of its pre-

decessors. The predecessor term is omitted if $\mathrm{Pred}_P(o)$ is empty. Operations already in progress retain their committed timings. Their completion times enter the evaluation without rescheduling those operations. For a feasible plan, this procedure yields the earliest start times consistent with its assignments, processing orders, and constraints imposed by the current system order.

The remaining makespan is

$$V(P; x_k) = \begin{cases} \max\limits_{o \in \mathcal{O}_k} f_o - t_k, & \mathcal{O}_k \neq \varnothing, \\ 0, & \mathcal{O}_k = \varnothing. \end{cases} \tag{8}$$

This cost measures the time needed to complete the currently observable unfinished work. It excludes unobserved future job arrivals and disruptions.

Accepting an improving candidate. A candidate that violates the scheduling constraints is rejected. When both plans admit a supported, feasible comparison, VPU computes the candidate's improvement as

$$\Delta_k = V(P_k; x_k) - V(Q_k; x_k). \tag{9}$$

The quality decision is

$$d_k = \begin{cases} \text{accept}, & \Delta_k > 0, \\ \text{no change}, & \Delta_k = 0, \\ \text{reject}, & \Delta_k < 0. \end{cases} \tag{10}$$

Thus, equal-cost and worse candidates leave the incumbent unchanged. When required information is missing, the comparison is unsupported and no replacement is made. The objective is the makespan of the complete observable continuation, so an accepted candidate does not need to reduce every individual job's completion time.

Committing the verified replacement. Acceptance alone does not deploy a candidate as the state may change while evaluation is in progress. VPU evaluates the detached candidate data outside the controller lock, then performs a final consistency check in a short critical section. This check verifies that the snapshot, incumbent identity and revision, operation scope, executed prefix, and candidate identity still match those used for verification. Upon a successful match, VPU atomically replaces the incumbent with the same candidate plan that was evaluated. Otherwise, the update is rejected.

Under valid public-state information, correct verification and synchronization, and unchanged candidate data, a committed replacement is feasible and strictly reduces the remaining makespan in the evaluated state. It is local to the evaluated continuation. Later observations and subsequent scheduling decisions can change both the scope and execution of that continuation.

4.3 Rollout-Based Action Arbitration

From plan replacement to action selection. VPU evaluates complete continuations at acceptance, but the scheduler executes one assignment at a time. A rule's next action can lose value as events and earlier assignments change the state (Ferreira et al., 2022). Thus, accepting a plan does not ensure that its next action remains preferable when later decisions follow the fast dispatching policy. RAA re-evaluates that action against fast-rule dispatch immediately before execution.

At step k, the synchronized plan proposes a_C, while the fast policy proposes $a_R = \pi_R(x_k, \mathcal{A}_k)$. In the PDR comparison, π_R is a classical priority rule (Holthaus & Rajendran, 2000). If a_C is unavailable or illegal, the controller uses a_R; otherwise, RAA clones the current state and starts one rollout with each action. Both rollouts use π_R for subsequent decisions, isolating the first action's effect under a shared fast-policy continuation.

Rollout comparison and fallback. Let J_C and J_R denote the terminal makespans of these two rollouts. The intended clone contract uses observable information, excludes unobserved future events, and leaves the live environment unchanged. Let $\mathcal{B}_k$ indicate that the candidate action is available and legal, both rollouts are supported, and both reach termination within a step limit H with finite costs. When RAA is enabled, the selected action is

$$a_k^\star = \begin{cases} a_C, & \mathcal{B}_k \wedge J_C < J_R - \epsilon_{\mathrm{RAA}}, \\ a_R, & \text{otherwise}, \end{cases} \tag{11}$$

where $\epsilon_{\mathrm{RAA}} \geq 0$ denotes the comparison tolerance. The candidate action is selected only when its rollout makespan is lower by more than this tolerance. Ties, worse outcomes, unavailable clones, unsupported comparisons, rollout failures, non-finite costs, and failure to terminate within H all select the fast-stream action.

Execution and subsequent decisions. After arbitration, the controller applies the selected action to the live environment once. A retained-plan proposal is recorded before arbitration, but execution is credited only after the environment receipt has been validated. A proposal, an arbitration decision, and a confirmed execution are therefore distinct events. At the next decision step, the retained plan is synchronized with the updated state and consulted again.

Collectively, the three stages operate on different objects. LSB selects the state reference for a request, VPU decides whether a complete candidate plan should replace the incumbent, and RAA selects the current action under a shared rollout continuation.

5 Experiments

5.1 Experimental Setup

We evaluate scheduling quality, model-token usage, and the contributions of LSB and VPU on 58 instances: 18 GEN-Bench-Small, 20 GEN-Bench-Normal, 10 Brandimarte, and 10 JMS. The 24 configurations include GP, six learning-based dispatchers, two LLM-based heuristic-search methods, LLM-Direct, TDSE, RACE, and AAI (Table 2). RACE serves as the control baseline.

Learning-based dispatchers are retrained on their target training distributions, whereas RACE and AAI do not require offline training. The two LLM-based heuristic-search adaptations use GPT-5-nano. LLM-Direct, TDSE, RACE, and AAI share three backbones: Qwen3.5-9B, Qwen3.5-27B, and Qwen3.5-35B. AAI is additionally evaluated with DeepSeek-V3.2 (DeepSeek-AI, 2025), GPT-5-nano (OpenAI, 2026), and Qwen3-Coder-30B.

To compare families with different makespan scales, we report the mean relative percentage deviation (RPD) and the mean within-instance rank. For configuration m on instance i,

$$\mathrm{RPD}_{i,m} = 100\frac{C_{i,m} - C_i^{\mathrm{ref}}}{C_i^{\mathrm{ref}}}, \tag{12}$$

where $C_{i,m}$ is terminal makespan and C_i^{ref} is the lowest makespan in the fixed non-ablation comparison pool, rather than a known optimum. Ranks are computed for all 24 displayed configurations, with ties receiving the same minimum rank. Both metrics are averaged equally across instances. We also report the mean model input plus output tokens per instance where comparable records are available.

5.2 Diagnostics of Slow-Stream Limitations

We first examine whether slow-stream usefulness is limited by state staleness and candidate quality. In the asynchronous diagnostic, no queue drops occur, but 66.7% of returned results are stale, including every delayed return (Table 1). Holding candidate bundles fixed across 348 matched blocks, removing delay increases normalized utility (zero-delay minus delayed) by 1.34 percentage points, with a 95% cluster-bootstrap interval of [0.45, 2.25] based on 10,000 resamples. Only 66.1% of zero-delay returns are deployed, distinguishing timely return from actual deployment. These findings motivate controlling state staleness and candidate quality; quality may also change after acceptance as the scheduler continues to act.

Table 1: Asynchronous-staleness diagnostic.

Quantity	**Result**
Queue drops	0/1,392
Stale returns	696/1,044
Stale delayed returns	696/696
Zero-delay deployments	230/348
Mean utility difference	+1.34 pp
95% bootstrap interval	[0.45, 2.25] pp

A separate diagnostic compares four selected candidates with a baseline continuation of makespan 61.589 from the same scheduling state (Table 3); positive gain denotes lower makespan. One can-

Table 2: Mean scheduling results across 58 instances (24 configurations).

Method	Source venue	Backbone	Makespan ↓	RPD (%) ↓	Rank ↓	Tokens (k) ↓
GP	GECCO '16 (Mei et al., 2016)	–	87.47	60.63	22.69	–
HMPSAC	J. Manuf. Syst. '25 (Ding et al., 2025)	–	65.21	16.26	14.07	–
IDDQN	Expert Syst. Appl. '25 (Wu et al., 2025)	–	63.80	15.00	13.86	–
DAN	IEEE TNNLS '24 (Wang et al., 2024)	–	66.73	17.72	15.34	–
PPO-OC	Comput. Oper. Res. '25 (Yuan et al., 2025)	–	65.35	18.34	16.24	–
DRL-CPO	Knowl. Based Syst. '26 (Nguyen et al., 2026)	–	69.58	22.19	17.50	–
MRGRL	Appl. Soft Comput. '26 (Pang et al., 2026)	–	59.28	7.90	7.74	–
LLM-HDfST	IEEE TEVC '26 (Yu et al., 2026)	–	66.79	16.91	14.81	–
DSevolve	arXiv '26 (Zhou et al., 2026)	–	67.16	16.50	14.69	–
LLM-Direct	ReflecSched: arXiv '25 (Cao & Yuan, 2025)	Qwen3.5-9B	65.23	17.51	15.48	154.3
		Qwen3.5-27B	64.60	14.82	13.67	146.9
		Qwen3.5-35B	64.18	16.46	14.28	146.0
TDSE	ReflecSched: arXiv '25 (Cao & Yuan, 2025)	Qwen3.5-9B	60.14	10.81	10.26	1,086.4
		Qwen3.5-27B	62.52	12.18	11.22	880.0
		Qwen3.5-35B	60.23	10.20	9.59	1,079.0
RACE	IJCAI '26 (Cao et al., 2026)	Qwen3.5-9B	61.17	12.33	11.29	56.5
		Qwen3.5-27B	60.67	12.22	11.36	71.2
		Qwen3.5-35B	60.61	12.95	12.00	121.7
AAI	This work	Qwen3.5-9B	60.22	7.55	6.05	11.8
		Qwen3.5-27B	59.50	7.91	6.16	8.8
		Qwen3.5-35B	59.88	7.48	6.14	11.8
		DeepSeek-V3.2	61.00	8.46	6.57	8.6
		GPT-5-nano	60.39	7.51	6.34	11.7
		Qwen3-Coder-30B	60.18	7.97	6.10	8.9

didate reduces makespan by 2.01%, while the other three increase it by 2.44%–6.01%. The wait-enabled simulator produces both an improvement and a regression. Thus, evaluating a candidate on the current state does not establish that it improves upon the incumbent, motivating VPU's explicit comparison of complete continuations. Even a favorable comparison at this point does not fix the quality of its future first actions, motivating RAA's repeated action-level check.

5.3 Scheduling Quality and Model-Token Usage

Table 2 shows that AAI achieves the three lowest mean RPDs and all six lowest mean ranks. Against RACE, its mean RPD falls by 4.78, 4.31, and 5.47 percentage points on the shared 9B, 27B, and 35B backbones; it also improves on TDSE for each matched backbone. MRGRL has the lowest raw mean makespan and the best non-AAI mean RPD, so AAI does not lead every aggregate metric.

Table 3: Four selected state-local candidate replays.

Simulator	Seed	Makespan	Gain (%)
Standard	7	63.089	-2.44
Wait	7	60.349	+2.01
Standard	17	65.029	-5.59
Wait	17	65.289	-6.01

AAI uses 8.6k–11.8k model tokens per instance, compared with 56.5k–121.7k for RACE. On the shared backbones, this represents token reductions of 79.2%, 87.6%, and 90.3%. Together with the RPD results, these matched comparisons show an improved quality–token trade-off.

5.4 Effect of Verified Plan Update

We compare RACE, VPU-only, and AAI on the same set of 58 instances and five seeds, holding shared execution support, including action arbitration, fixed. For each instance–seed pair, we compute the relative makespan reduction against RACE, average across available seeds within each instance, and then summarize across instances.

Table 4: Terminal makespan reductions relative to RACE (58 instances).

Configuration	Mean	Median	W/T/L
RACE	0.00	0.00	-
VPU only	10.33	8.99	52/2/4
AAI	10.74	10.16	51/2/5

VPU alone reduces makespan by 10.33% on average and improves 52 of 58 instances (Table 4); W/T/L counts improved, tied, and worsened instance means. AAI yields a slightly larger mean reduction of 10.74%, but improves 51 instances and worsens five. These results indicate that the VPU contributes to the observed improvements in benchmark gain.

Figure 4 shows positive mean reductions in all four benchmark families: 13.83% in Brandimarte, 9.88% in GEN-Bench-Normal, 10.15% in GEN-Bench-Small, and 8.05% in JMS. However, the effects at the instance level range from approximately -5% to 39%, with four regressions in the two GEN-Bench families. This heterogeneity separates VPU's local acceptance condition from terminal scheduling quality. An accepted replacement improves the observable continuation on the evaluated state, but later observations and action choices can change the actually executed continuation. The results therefore support an average terminal benefit within the evaluated benchmarks.

Figure 4 shows positive mean reductions in all four benchmark families, ranging from 8.05% in JMS to 13.83% in Brandimarte. Instance-level effects nevertheless range from approximately -5% to 39%, including four regressions in the two GEN-Bench families. VPU accepts a replacement based on the evaluated continuation; later observations and actions can change the executed schedule, so local acceptance does not guarantee a terminal improvement.

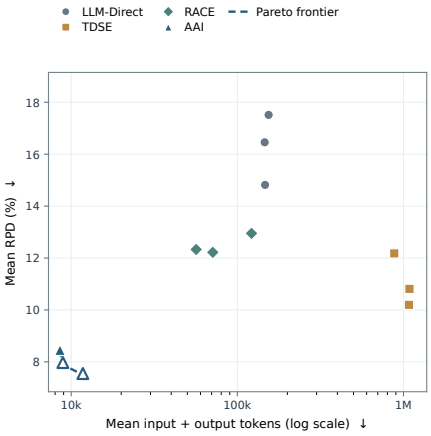


Figure 3: Scheduling quality versus model-token usage.

5.5 Comparison with PDR-Only Dispatching

We examine whether a classical PDR alone can explain the scheduling quality obtained with AAI. Table 5 highlights the best and worst reported RPD and the two lowest PDR-only RPD; Appendix A gives all supplied values.

The lowest reported PDR-only RPD is 9.7815 (MOPNR:SPT), above all six model-labelled AAI RPD (8.0294–9.0099). This pattern is consistent with useful decisions being retained and selected during execution rather than simply running one fixed classical rule.

5.6 Timing Dependence of Late State Binding

Temporal-Fork compares trigger-time (T) and claim-time (C) binding by placing a critical arrival before claim; T/C denotes the mean paired cost ratio. Within each timing condition, both rules use the same constructed scheduling case and differ only in the moment at which they select the state reference. Three size strata, eight parameter seeds, two arrival timings, and two binding rules yield 96 observations with 24 matched size–seed pairs per timing condition.

Table 5: Selected AAI and PDR-only RPD.

Family	Model or rule pair	RPD(%) ↓
AAI	Qwen3.5-35B	8.0294
	DeepSeek-V3.2	9.0099
PDR only	MOPNR:SPT	9.7815
	MWKR:SPT	10.1954

When the critical arrival becomes observable between trigger and claim, claim-time binding reduces mean synthetic makespan from 22.03 to 16.41 and wins all 24 matched pairs (Table 6). The effect holds across all three sizes, with mean paired cost ratios of 1.338–1.349. Collectively, the benchmark and timing results support a conditional contribution from LSB.

Table 6: Temporal-Fork synthetic makespans by binding time.

Stratum	T	C	T/C
Overall	22.03	16.41	1.342
Small	21.75	16.13	1.349
Medium	22.11	16.49	1.341
Large	22.24	16.61	1.338

6 Conclusion

AAI coordinates LLM-generated rules with ongoing dispatching through claim-time state binding, verified plan replacement, and action-level arbitration. Across 58 instances, it lowers mean RPD by about five percentage points versus RACE on three shared backbones while cutting model-token usage by up to about 90%. The component and timing results attribute the benchmark gain chiefly to VPU and identify when LSB benefits from newly observed information.

States can change during LLM generation, and later events can reverse an improvement verified on the current state. Future work will adapt refresh and arbitration to changing event rates and measure end-to-end cost under inference delays.

AI USE STATEMENT

We used generative AI tools to assist with implementing parts of the methodological code, polishing manuscript language, organizing the structure and presentation of the paper, and identifying and surveying relevant literature. All AI-assisted code was reviewed and checked by the authors. The authors also critically reviewed AI-assisted writing and literature suggestions and take full responsibility for the final manuscript, including its claims, analyses, code, figures, and references.

REPRODUCIBILITY STATEMENT

The source code is available at https://anonymous.4open.science/r/ICLR39029-B3B8

REFERENCES

Shijie Cao and Yuan Yuan. Reflecsched: Solving dynamic flexible job-shop scheduling via llm-powered hierarchical reflection. *CoRR*, abs/2508.01724, 2025. doi: 10.48550/ARXIV.2508.01724. URL https://doi.org/10.48550/arXiv.2508.01724.

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 and Evolutionary Computation*, 83:101419, 2023. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2023.101419. URL https://www.sciencedirect.com/science/article/pii/S221065022300192X.

Shijie Cao, Yuan Yuan, and Jing Liu. Harmonizing real-time constraints and long-horizon reasoning: An asynchronous agentic framework for dynamic scheduling. In Diego Calvanese (ed.), *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-26*, pp. 6146–6155. International Joint Conferences on Artificial Intelligence Organization, 8 2026. doi: 10.24963/ijcai.2026/684. URL https://doi.org/10.24963/ijcai.2026/684. Main Track.

Inguk Choi, Woo-Jin Shin, Sang-Hyun Cho, and Hyun-Jung Kim. Towards generalizable multi-policy optimization with self-evolution for job scheduling. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 24569–24616. Curran Associates, Inc., 2025. doi: 10.52202/085713-0828. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/2330c4be5e12c504705ba1db8a507966-Paper-Conference.pdf.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://doi.org/10.48550/arXiv.2512.02556.

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. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2025.03.019. URL https://www.sciencedirect.com/science/article/pii/S0278612525000809.

Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. *Omega*, 111:102643, 2022. ISSN 0305-0483. doi: https://doi.org/10.1016/j.omega.2022.102643. URL https://www.sciencedirect.com/science/article/pii/S0305048322000512.

Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant RL populations for combinatorial optimization. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *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*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/97b983c974551153d20ddfabb62a5203-Abstract-Conference.html.

Oliver Holthaus and Chandrasekharan Rajendran. Efficient jobshop dispatching rules: Further developments. *Production Planning & Control*, 11(2):171–178, 2000. doi: 10.1080/095372800232379. URL https://doi.org/10.1080/095372800232379.

Sirui Li, Wenbin Ouyang, Yining Ma, and Cathy Wu. Learning-guided rolling horizon optimization for long-horizon flexible job-shop scheduling. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 1724–1768, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/052e2404709e8df9dc38fbc7bfec5b80-Paper-Conference.pdf.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

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 (eds.), *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pp. 365–372. ACM, 2016. doi: 10.1145/2908812.2908822. URL https://doi.org/10.1145/2908812.2908822.

Vu-Tien Nguyen, Su Nguyen, and Van-Hop Nguyen. Deep reinforcement learning to select efficient portfolios of search heuristics for solving dynamic job shop scheduling. *Knowl. Based Syst.*, 348:116417, 2026. doi: 10.1016/J.KNOSYS.2026.116417. URL https://doi.org/10.1016/j.knosys.2026.116417.

OpenAI. OpenAI GPT-5 System Card, 2026. URL https://doi.org/10.48550/arXiv.2601.03267.

Zhi Pang, Xinyu Li, Liang Gao, Xuxia Zhang, and Jun Wu. Multi-relational graph reinforcement learning for dynamic flexible job-shop scheduling under machine breakdowns. *Appl. Soft Comput.*, 190:114587, 2026. doi: 10.1016/J.ASOC.2026.114587. URL https://doi.org/10.1016/j.asoc.2026.114587.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, Jan 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Generalizable heuristic generation through LLMs with meta-optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tIQZ7pVN6S.

Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. Flexible job shop scheduling via dual attention network-based reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3091–3102, 2024. doi: 10.1109/TNNLS.2023.3306421.

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 Systems with Applications*, 288:128280, 2025. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2025.128280. URL https://www.sciencedirect.com/science/article/pii/S0957417425018998.

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. *Artificial Intelligence Review*, 58(6):160, Mar 2025. ISSN 1573-7462. doi: 10.1007/s10462-024-11059-9. URL https://doi.org/10.1007/s10462-024-11059-9.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608. Curran Associates, Inc., 2024. doi: 10.52202/079017-1381. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/4ced59d480e07d290b6f29fc8798f195-Paper-Conference.pdf.

Fei Yu, Liang Gao, Xinyu Li, Chao Lu, and Qihao Liu. Automated scheduling heuristic generation and evaluation via large language model. *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2026. doi: 10.1109/TEVC.2026.3655772.

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. *Computers & Operations Research*, 183:107149, 2025. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2025.107149. URL https://www.sciencedirect.com/science/article/pii/S0305054825001777.

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 (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/11958dfee29b6709f48a9ba0387a2431-Abstract.html.

Cong Zhang, Zhiguang Cao, Yaoxin Wu, Wen Song, and Jing Sun. Learning topological representations with bidirectional graph attention network for solving job shop scheduling problem. In Negar Kiyavash and Joris M. Mooij (eds.), *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*, volume 244 of *Proceedings of Machine Learning Research*, pp. 4192–4208. PMLR, 15–19 Jul 2024. URL https://proceedings.mlr.press/v244/zhang24c.html.

Junjie Zhang, Canhui Luo, Zhouxing Su, Qingyun Zhang, Zhipeng Lü, Junwen Ding, and Yan Jin. Ns4s: Neighborhood search for scheduling problems via large language models. In James Kwok (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 8687–8695. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/966. URL https://doi.org/10.24963/ijcai.2025/966. Main Track.

XinLei Zhou, Jin Huang, Jie Yang, Xinyu Li, and Liang Gao. Dsevolve: Enabling real-time adaptive scheduling on dynamic flexible job shop with llm-evolved heuristic portfolios, 2026. URL https://arxiv.org/abs/2603.27628.

A Complete PDR-Only Comparison

Table 7 reports all six model-labelled AAI scores and all 24 supplied PDR-only rule-pair scores. The score definition and evaluation matching remain unconfirmed, as discussed in Section 5.5.

B Baseline Settings

Table 8 lists the method-specific settings that complement the shared protocol in the main text. Here O_{sum} denotes the source protocol's operation-count quantity, and offline units retain their reported unit.

C Paired Comparisons

Table 9 compares the 24 primary configurations using AAI–Qwen3.5-27B as the fixed reference. The reported, unadjusted p values use a one-sided upper-tail normal approximation to the Wilcoxon

Table 7: Complete reported AAI and PDR-only RPD (lower is better, pp).

Family	Model or rule pair	RPD (%) ↓	Family	Model or rule pair	RPD (%) ↓
AAI	Qwen3.5-9B	8.1020	PDR only	LWKR:LIT	21.0305
	Qwen3.5-27B	8.4561		LWKR:LWL	23.6586
	Qwen3.5-35B	8.0294		LWKR:SPT	19.1737
	DeepSeek-V3.2	9.0099		MOPNR:LIT	11.4920
	GPT-5-nano	8.0617		MOPNR:LWL	11.6613
	Qwen3-Coder-30B	8.5248		MOPNR:SPT	9.7815
PDR only	SPT:LIT	14.1306	PDR only	LOPNR:LIT	22.5606
	SPT:LWL	16.2595		LOPNR:LWL	22.8941
	SPT:SPT	12.2038		LOPNR:SPT	17.3507
	LPT:LIT	21.3389		FIFO:LIT	18.1536
	LPT:LWL	21.7018		FIFO:LWL	18.0925
	LPT:SPT	17.9469		FIFO:SPT	14.7394
	MWKR:LIT	11.0308		LIFO:LIT	13.4141
	MWKR:LWL	11.4104		LIFO:LWL	14.9882
	MWKR:SPT	10.1954		LIFO:SPT	12.8441

Table 8: Baseline training and evaluation settings.

Method	Training and evaluation settings
GP	Source-recommended configuration; individual hyperparameters are not specified in the available records.
HMPSAC	1,000 episodes, at most 10,000 decision steps; learning rate 3×10^{-4}, $\gamma = 0.99$, value/entropy coefficients 0.5/0.01.
IDDQN	2,000 episodes; learning rate 0.01, $\gamma = 0.9$, batch O_{sum}, buffer $10O_{\text{sum}}$, prioritized-replay $\alpha/\beta = 0.6/0.4$, target update every O_{sum} steps.
DAN	Source training protocol; learning rate 3×10^{-4}, clipping 0.2, value/entropy coefficients 0.5/0.01, GAE 0.98, 4 epochs.
PPO-OC	Source training protocol; learning rate 3×10^{-4}, clipping 0.2, value/entropy coefficients 0.5/0.01, GAE 0.95, 10 epochs.
DRL-CPO, MR-GRL	Adapted RL; 20 training instances, 500 offline units per instance, seed 20260905, 200-ms online decisions, CUDA; evaluation on the 58 shared instances.
LLM-HDfST, DSe-volve	Adapted LLM search; GPT-5-nano, 20 sequential training instances, 500 offline units per instance, seed 20260905, 200-ms online decisions, CPU; evaluation on the 58 shared instances.
LLM-Direct	No offline training; Qwen3.5-9B/27B/35B; shared executable-action interface.
TDSE	Rollout factor 24, maximum depth 6, 3 iterations; decoding temperature/top-p 0.2/0.8, top-k 20, maximum 8,192 tokens, reflective non-thinking, search temperature 0.8.
RACE, AAI	No offline training; model-specific evaluation on the 58 shared instances. RACE disables LSB and VPU; AAI enables both.

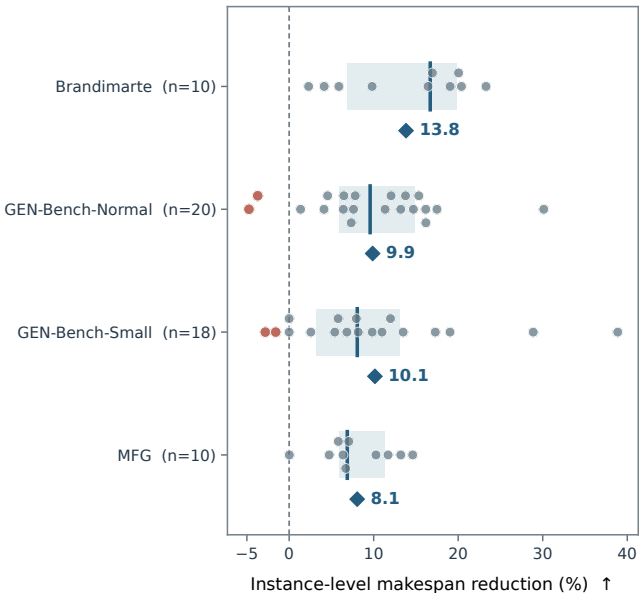


Figure 4: Instance-level VPU reductions relative to RACE.

signed-rank statistic, excluding zero differences and averaging ranks for equal absolute differences. W/T/L counts lower/equal/higher RPD for the fixed reference than for each row; the dash denotes self-comparison.

Table 9: Paired comparisons with AAI–Qwen3.5-27B (58 instances).

Method	**Backbone**	**Reference W/T/L**	p
GP	–	56/0/2	2.28×10^{-11}
HMPSAC	–	44/1/13	3.78×10^{-6}
IDDQN	–	44/2/12	2.99×10^{-6}
DAN	–	46/1/11	3.49×10^{-7}
PPO-OC	–	48/2/8	2.31×10^{-8}
DRL-CPO	–	48/1/9	2.21×10^{-9}
MRGRL	–	26/3/29	0.553
LLM-HDfST	–	44/2/12	6.26×10^{-8}
DSevolve	–	45/4/9	5.86×10^{-7}
LLM-Direct	Qwen3.5-9B	47/2/9	8.67×10^{-9}
	Qwen3.5-27B	41/2/15	4.55×10^{-6}
	Qwen3.5-35B	43/0/15	2.10×10^{-7}
TDSE	Qwen3.5-9B	36/1/21	0.013
	Qwen3.5-27B	37/3/18	2.38×10^{-4}
	Qwen3.5-35B	35/2/21	0.022
RACE	Qwen3.5-9B	42/0/16	6.84×10^{-4}
	Qwen3.5-27B	42/1/15	4.06×10^{-4}
	Qwen3.5-35B	43/0/15	1.98×10^{-4}
AAI	Qwen3.5-9B	9/40/9	0.491
	Qwen3.5-35B	12/35/11	0.608
	DeepSeek-V3.2	11/39/8	0.210
	GPT-5-nano	13/32/13	0.716
	Qwen3-Coder-30B	9/41/8	0.344
AAI	Qwen3.5-27B	–	1.000

D Component Results and Data Coverage

The component experiment includes RACE, LSB only, VPU only, and AAI, with 1,160 observations. Tables 10 and 11 provide the family- and instance-level values underlying the main-text summaries. In Table 10, n counts instances, RACE and VPU report mean makespan, and reduction averages paired percentage changes over seeds and then instances; W/T/L counts improved/tied/worsened instance means. Positive entries in Table 11 indicate improvement compared with RACE.

Table 10: Family-level VPU effects relative to RACE.

Family	n	**RACE**	**VPU**	**Reduction** (%)	**W/T/L**
Brandimarte	10	221.63	188.21	13.83	10/0/0
GEN-Bench-Normal	20	55.05	49.35	9.88	18/0/2
GEN-Bench-Small	18	16.89	15.00	10.15	14/2/2
JMS (MFG)	10	40.77	37.41	8.05	10/0/0

Table 11: Instance-level makespan reductions relative to RACE (%).

Family	**Instance**	**VPU only**	**AAI**	**Instance**	**VPU only**	**AAI**
Brandimarte	Mk01	9.81	14.96	Mk02	23.29	24.71
	Mk03	16.43	16.51	Mk04	4.14	2.95
	Mk05	20.38	20.11	Mk06	5.89	5.04
	Mk07	20.04	23.95	Mk08	19.04	20.60
	Mk09	2.30	1.25	Mk10	16.95	18.67
Gen-Bench Normal	instance_1	11.35	10.50	instance_2	-3.72	−3.72
	instance_3	4.55	4.57	instance_4	15.34	15.34
	instance_5	4.12	7.02	instance_6	14.70	14.52
	instance_7	1.35	3.88	instance_8	13.77	13.04
	instance_9	6.41	6.41	instance_10	17.49	16.70
	instance_11	16.18	13.98	instance_12	7.81	8.31
	instance_13	6.47	6.47	instance_14	30.12	30.12
	instance_15	13.20	14.23	instance_16	7.34	13.82
	instance_17	-4.76	−4.76	instance_18	7.61	8.77
	instance_19	12.06	13.20	instance_20	16.16	18.01
Gen-Bench Small	instance_1	7.95	7.95	instance_2	17.29	17.29
	instance_3	38.87	38.87	instance_4	5.40	5.40
	instance_5	10.99	12.75	instance_6	6.83	6.83
	instance_7	-2.82	−2.82	instance_8	-1.59	−1.59
	instance_9	2.57	2.57	instance_10	0.00	0.00
	instance_11	13.49	13.49	instance_12	5.80	5.80
	instance_13	9.83	9.83	instance_14	8.17	8.17
	instance_15	11.99	11.99	instance_16	0.00	0.00
	instance_17	19.03	19.03	instance_18	28.88	28.88
JMS-Bench	JMS 01	11.71	11.78	JMS 02	5.82	6.97
	JMS 03	14.61	13.59	04	7.06	7.75
	JMS 05	0.03	−1.08	JMS 06	10.27	10.81
	JMS 07	6.37	6.06	JMS 08	6.70	7.93
	JMS 09	4.74	1.97	JMS 10	13.21	13.31

E Additional Diagnostic Details

Temporal-Fork construction. The parameter seeds are $\{101, 103, 107, 109, 113, 127, 131, 137\}$. The generator ranks registered job weights, assigning weight 1000.0 to the critical job when visible; synthetic makespan is computed from constructed durations, the selected action, and distractor terms. The experiment makes no LLM, provider, or simulator calls. In the complementary condition where the critical job becomes observable only after claim, the two binding rules tie in all 24 pairs,

with mean synthetic makespan 16.41. This control tests the timing boundary of the construction, not correspondence between a coordinator snapshot and an actual provider input.

Diagnostic scope. The asynchronous diagnostic measures a utility contrast under fixed candidate bundles. The four state-local replays use one instance, seeds 7 and 17, and standard and wait-enabled JMSSim simulators.