CausaSched: A Causal-Guided Deep Reinforcement Learning Hyper-Heuristic for Improving Multiple Shop Scheduling Problems

Anonymous authors
Paper under double-blind review

Abstract

Traditional constructive heuristics and most deep reinforcement learning (DRL) scheduling methods generate schedules from scratch, whereas recent DRL-based improvement methods start from existing feasible solutions and further improve their quality through successive neighborhood moves. Because schedule improvement concerns how changes in controllable decisions affect the makespan, this process can be modeled from a causal-intervention perspective. Based on this observation, we propose CausaSched, a causal-guided DRL hyper-heuristic framework for multiple scheduling problems. Causal reasoning starts from scheduling bottlenecks, traces how delays arise through structural dependencies, and determines where subsequent schedule modifications should be applied. Achieving this goal raises three key questions: how to efficiently search for intervention targets that can reduce makespan, how to match them with neighborhood operators satisfying problem-specific constraints, and how to jointly train both decision networks to improve search performance. CausaSched exploits job-precedence and machine-order dependencies shared across scheduling problems: a causal reasoning network learns where to intervene, and an operator selection network determines the neighborhood move. We further propose Causal Hierarchical Group Relative Policy Optimization (CH-GRPO), which jointly trains both networks using multiple multi-step search trajectories and expands the neighborhoods of high-quality states. Experiments on public benchmark instances and mixed synthetic instances show that CausaSched significantly outperforms constructive DRL methods and improvement heuristics.

1 Introduction

Shop scheduling determines the processing order and execution of jobs over limited production resources, directly affecting production cycles, resource utilization, and on-time delivery. The job shop scheduling problem (JSP), flow shop scheduling problem (FSP), flexible job shop scheduling problem (FJSP), and hybrid flow shop scheduling problem (HFSP) are four representative variants characterized by different routing structures, stage configurations, and degrees of resource flexibility. Exact algorithms, heuristics, and metaheuristics have achieved substantial progress on these problems (Carlier & Néron, 2000; Nawaz et al., 1983; Ruiz & Stützle, 2007). Nevertheless, the combinatorial explosion caused by increasing problem scales has motivated the use of deep reinforcement learning (DRL) to learn scheduling policies that can solve instances rapidly and generalize across instances (Bengio et al., 2021; Zhang et al., 2020; Park et al., 2021; Song et al., 2023; Wang et al., 2024).

Many DRL approaches to scheduling follow a constructive paradigm, in which operations and resources are assigned sequentially from an empty state until a complete schedule is obtained (Zhang et al., 2020; Song et al., 2023; Wang et al., 2024). These methods can produce feasible schedules in a single pass at relatively low inference cost, but they generally cannot revise earlier decisions and may therefore be constrained by mistakes made at the beginning of construction. In contrast, improvement methods start from a complete feasible schedule and search for better solutions through a sequence of neighborhood moves (Falkner et al., 2022; Zhang et al., 2024; Wang et al., 2025). They can revise existing decisions, extend the search according to the available computational budget, and

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.

accept initial solutions produced by different solvers. Improvement learning therefore offers a more flexible way to further reduce the makespan and constitutes the basic paradigm considered in this work.

Recent learning-based improvement methods have demonstrated that DRL can guide neighborhood search over complete feasible schedules and progressively improve solution quality (Zhang et al., 2024; Wang et al., 2025). Representative approaches select candidate operations from critical paths or critical blocks (Zhang et al., 2007) and reconstruct the schedule within predefined neighborhoods. We also improve complete solutions, but focus on a different question: which sources of delay in the current schedule have affected the makespan, and how can scheduling interventions attenuate their effects? Critical-path-guided methods can discover effective modifications by repeatedly applying neighborhood search. We instead seek to analyze the factors affecting the makespan before applying an operator, and subsequently determine both the operations to intervene on and the corresponding neighborhood operators. This consideration motivates us to introduce causal reasoning into learning-based schedule improvement.

Operation sequencing, resource occupancy, and temporal coordination are interdependent in a schedule. Local delays may propagate through technological constraints and resource relations and eventually affect the makespan, suggesting a new direction for schedule improvement. Inspired by causal root cause analysis (Budhathoki et al., 2021; 2022; Han et al., 2025), we use a causal graph to characterize the dependencies among scheduling states. Makespan-related bottlenecks are treated as consequences of delay propagation, and their propagation paths are traced backward to candidate operations that may have produced these effects. The model further combines evidence from multiple influence paths with realized scheduling returns to estimate the intervention value of each candidate and inform subsequent neighborhood-operator selection. Because this process concerns how delays propagate through scheduling dependencies rather than a machine environment specific to one problem, the same causal tracing and intervention mechanism can be applied to JSP, FSP, FJSP, and HFSP.

A shared causal reasoning mechanism, however, does not allow the constraints specific to the four scheduling families to be ignored. Job-precedence and machine-order dependencies provide a common basis for causal analysis, while candidate interventions must satisfy problem-specific routing, resource eligibility, and operation constraints. Moreover, multiple sources of delay may be connected to the same upstream operation through different paths, and a selected operation must still be translated into a concrete modification that satisfies the relevant constraints. Because the intervention target and intervention operator jointly determine the search outcome, the two decisions must also be learned under a consistent optimization objective.

We propose CausaSched, a causal-guided hyper-heuristic framework for improving multiple families of shop schedules. It combines causal tracing with constraint-aware interventions through shared job and machine dependencies. We design a causal reasoning network and an operator selection network: learned causal tracing (LCT) locates intervention targets, while single and compositional operators (CO) produce schedule modifications, coordinating target and operator decisions. CH-GRPO jointly trains the two policies by combining step-level and trajectory-level feedback. Experiments on public benchmarks and mixed instances show that CausaSched outperforms constructive DRL and improvement heuristics in most size groups; ablations demonstrate each component's contribution.

2 Related Work

2.1 DRL-Based Schedule Construction

DRL-based scheduling methods formulate construction as sequential decision-making and learn dispatching and resource-assignment policies (Zhang et al., 2020; Park et al., 2021; Song et al., 2023; Wang et al., 2024). L2D represents a partial JSP schedule as a disjunctive graph and uses a graph neural network to select the next operation (Zhang et al., 2020). For FJSP, HGNN explicitly models operation–machine relations, while DANIEL uses dual attention to represent operations and competing machines (Song et al., 2023; Wang et al., 2024). DRL has also been applied to dynamic permutation flowshops (Yang et al., 2021) and distributed hybrid flowshops (Di et al., 2024). These methods can rapidly generate feasible schedules and generalize across instance sizes; their outputs can also serve as initial solutions for subsequent improvement.

2.2 DRL-Based Schedule Improvement

Unlike constructive methods, DRL-based improvement methods treat a complete schedule as the search state and improve it through successive neighborhood moves (Falkner et al., 2022; Zhang et al., 2024; Wang et al., 2025). NeuRewriter separates region selection from rewriting-rule selection (Chen & Tian, 2019), while NeuroLS learns to control move acceptance, neighborhood selection, and perturbation (Falkner et al., 2022). Zhang et al. (2024) represent a JSP solution with a complete disjunctive graph, use a graph neural network to select local moves, and introduce parallel message passing to accelerate candidate evaluation. For flexible machine assignment in FJSP, MIStar uses a heterogeneous disjunctive graph to represent sequencing and machine-selection decisions, augments the state with search memory, and explores neighborhoods using a parallel greedy policy (Wang et al., 2025). Following the hyper-heuristic paradigm (Burke et al., 2013), MGRO and related approaches learn to select among predefined neighborhood operators and combine reinforcement learning with iterative greedy or other metaheuristic search procedures (Ni et al., 2021).

These studies show that learned policies can effectively control neighborhood search over complete solutions. However, representative methods primarily address a single scheduling family. To address this limitation, CausaSched exploits shared scheduling dependencies and constraint-aware operations to support JSP, FSP, FJSP, and HFSP.

2.3 Causal Reasoning and Root Cause Analysis

Causal root cause analysis uses dependencies among variables to identify sources of downstream changes. Existing approaches attribute distribution shifts to changes in causal mechanisms (Budhathoki et al., 2021) or quantify upstream contributions to an outlier using a given causal graph and functional causal model (Budhathoki et al., 2022). AERCA combines Granger causal discovery with root cause analysis, formulates anomalies as interventions on exogenous variables in a structural causal model, and localizes anomaly sources using learned temporal causal relations (Han et al., 2025). This line of work shows that causal structure can connect observed downstream outcomes to upstream factors that may have produced them.

CausaSched adopts the idea of tracing sources along influence relations but targets schedule optimization rather than anomaly diagnosis. Scheduling constraints provide the structural relations used to organize candidate interventions, while realized makespan returns supervise where to modify the schedule and which operator to apply. Causal reasoning therefore becomes part of a hyper-heuristic policy that directly guides search in the solution space.

3 Preliminaries and Problem Formulation

3.1 Scheduling Optimization Problem

We improve complete feasible schedules for JSP, FSP, FJSP, and HFSP. An instance $\mathcal{I} = (\mathcal{J}, \mathcal{O}, \mathcal{M}, \mathcal{E}^P)$ comprises jobs, operations, machines, and job-precedence relations. Each operation o has eligible modes $\mathcal{Q}_o$, specifying a machine $m(q)$ and processing time $p_{o,q}$. A schedule selects a mode q_o and start time s_o, giving $c_o = s_o + p_{o,q_o}$. Feasibility requires $s_v \geq c_u$ for $(u, v) \in \mathcal{E}^P$ and no overlap on a unit-capacity machine. The objective is to minimize $C_{\max}(S) = \max_o c_o$.

This formulation retains family-specific constraints: JSP fixes machines along job-specific routes; FSP uses a common stage order with one machine per stage; FJSP permits eligible-machine selection; HFSP combines a common stage order with parallel-machine selection.

3.2 Scheduling Dependencies

All four families share job-precedence and machine-order relations in a complete schedule, following the disjunctive-graph formulation. We write $\mathcal{G}(S) = (\mathcal{O} \cup \{v_s, v_t\}, \mathcal{E}^P \cup \mathcal{E}^R)$, where v_s, v_t are source/sink nodes, $\mathcal{E}^P$ includes their connections, and $\mathcal{E}^R$ records machine orders. Figure 1(a) shows job precedence and eligible machines; (b) resolves assignments and orders.

We examine delays associated with operation waiting, machine idle gaps (Maassen et al., 2020), and workload imbalance (Chen & Chen, 2009), identifying scheduling blocks whose effects reach

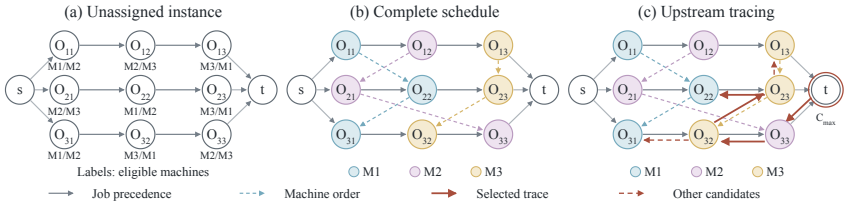


Figure 1: Unified scheduling representation and upstream tracing.

the makespan through tight job- and machine-dependency chains, where each predecessor finishes exactly when its successor starts. This can be interpreted as an initial search for promising blocks on makespan-determining critical paths (Zhang et al., 2007), followed by upstream tracing along job and machine predecessors. The approach focuses on critical-path delay blocks while extending interventions to associated upstream branches (Appendix B.1). Figure 1(c) illustrates this tracing: solid red edges mark the selected path and dashed edges indicate other upstream candidates.

3.3 Causal Reasoning

We combine root cause localization and counterfactual evaluation to identify interventions that can reduce makespan. Localization follows the scheduling dependencies to identify candidate intervention targets (Han et al., 2025). Counterfactual evaluation changes selected operations' machine assignments or processing orders while holding instance conditions and unmodified decisions fixed; fixed-decision replay checks feasibility and computes makespan. Makespan changes guide target and operator learning, and reconstructed schedules enter subsequent search while retaining the best solution found.

4 Method: CausaSched

Starting from a feasible schedule, CausaSched encodes its state, selects intervention targets, and chooses neighborhood moves, with Intervention Memory supporting both policies (Figure 2). Each move reconstructs the schedule; search retains the best solution. CH-GRPO trains both policies with step and trajectory feedback. Pretraining and LLM-assisted data generation are detailed in Appendix B.4.

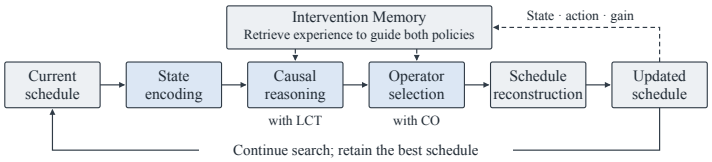


Figure 2: Overview of CausaSched.

4.1 Markov Decision Process

We formulate schedule improvement as a Markov decision process (MDP).

State. Let $s_t = (G_t, \mathbf{g}_t, \{(\mathbf{x}_u, \mathbf{m}_u)\}_{u \in \mathcal{U}_t})$, where G_t is the scheduling dependency graph, $\mathbf{g}_t$ describes makespan and search progress, $\mathcal{U}_t$ is the candidate operation set, and $\mathbf{x}_u, \mathbf{m}_u$ are operation u's local features and historical intervention information.

Action. The causal policy selects the target set $U_t \subseteq \mathcal{U}_t$, $|U_t| \leq K$, where K is the per-step target limit. The operator policy constructs the target–operator pair set $\mathcal{A}_t = \mathcal{A}_t(U_t)$, including machine rerouting, sequence swap, sequence insertion, and permitted joint modifications. Executing $a_t \in \mathcal{A}_t$ through fixed-decision replay yields S_{t+1}.

Reward. Let $C_{i,t} = C_{\max}(S_{i,t})$ and $\Delta C_{i,t} = C_{i,t} - C_{i,t+1}$. The step reward is $r_{i,t} = [\Delta C_{i,t}]_+ - \lambda_{\text{reg}}[-\Delta C_{i,t}]_+$, where $[x]_+ = \max(x, 0)$. We also evaluate each trajectory by $r_i^{\text{traj}} = (C_0 - C_i^\star)/C_0$,

where $C_i^\star = \min_{0 \le t \le T_i} C_{i,t}$ includes the shared start C_0. These signals measure immediate changes and best improvement within budget, respectively.

4.2 Causal Reasoning

After state encoding, the causal reasoning network identifies root nodes with high optimization potential from the candidate set $\mathcal{U}_t$ and produces the intervention target set U_t. As illustrated in Figure 4, the network consists of two parallel branches: relational context encoding and learned causal tracing. The former captures the association between each candidate operation and the current delay context, while the latter models how delay responsibility propagates and aggregates upstream along scheduling dependencies. Their outputs determine candidate scores through Root Scoring.

State encoding. State encoding comprises graph encoding, sequence encoding and fusion, and reverse-dependency encoding. A relation-aware GAT (Veličković et al., 2018) encodes heterogeneous scheduling graphs, while a local sequence Transformer (Vaswani et al., 2017) extracts temporal features in parallel; their outputs are aligned by operation and fused through a gate. The fused representations enter a reverse-dependency encoder, which aggregates information along job and machine predecessors conditioned on the delay context, yielding candidate operation representations $\mathbf{e}_u$ (encoding and Memory details in Appendix B.2).

Relational context encoding. Dependency Aggregation combines operation representations $\mathbf{e}_u$ with relation features to obtain the operation-level prior z_u^{prior}. The candidate representation is $\mathbf{h}_u = \text{MLP}([\mathbf{e}_u; z_u^{\text{prior}}; \mathbf{x}_u; \mathbf{m}_u])$, incorporating local state and historical experience. Multi-head cross-attention $\text{Attn}_{\text{delay}}$ takes $\mathbf{h}_u$ as the query, aggregates the relation features $\mathbf{r}_{u,k}$ between the candidate and each delay block, and produces $\mathbf{h}'_u$ through a residual connection. The candidate-set Transformer jointly processes these tokens and the state token $\mathbf{h}_s$ derived from $\mathbf{g}_t$, as $[\tilde{\mathbf{h}}_s; \{\tilde{\mathbf{h}}_u\}_{u \in \mathcal{U}_t}] = \text{Trans}_{\text{root}}([\mathbf{h}_s; \{\mathbf{h}'_u\}_{u \in \mathcal{U}_t}])$. Set-level self-attention yields contextualized candidate representations $\tilde{\mathbf{h}}_u$ for root scoring.

Learned causal tracing (LCT). As illustrated in Figure 3, the second branch propagates and aggregates delay responsibility upstream from operations associated with multiple delay factors along scheduling dependencies. For each delay block b, the initial seed mass is determined by the block weight ρ_b and seed weight $\eta_{b,v}$, i.e., $q^{(0)}(v) = \sum_b \rho_b \eta_{b,v} \mathbb{I}[v \in V_b]$. At tracing layer ℓ, a scoring network with L hidden layers predicts normalized transition weights $T_\theta^{(\ell)}(u|v)$ over the node itself and its legal predecessors, from node representations and the current search context (Appendix B.3). Responsibility propagates as

$$q^{(\ell+1)}(u) = \sum_{v : u \in \{v\} \cup \text{Pa}(v)} q^{(\ell)}(v) T_\theta^{(\ell)}(u|v). \tag{1}$$

Mass not transferred out is retained at the current node. Tracing follows legal predecessors without a fixed depth cap, subject to per-layer width limits. Shared representation and scoring modules produce a score s_ℓ for each expanded layer, normalized as $\beta_\ell = \exp(s_\ell) / \sum_{k=0}^{D} \exp(s_k)$, where D is the tracing depth reached at the current decision. Aggregating these layers gives

$$p_{\text{tr}}(u) = \frac{\epsilon + \sum_\ell \beta_\ell q^{(\ell)}(u)}{\sum_{w \in \mathcal{U}_t} [\epsilon + \sum_\ell \beta_\ell q^{(\ell)}(w)]}. \tag{2}$$

The sums cover $0, \ldots, D$; the $\ell = 0$ term retains direct selection of delay-related operations. We use $\epsilon = 10^{-6}$.

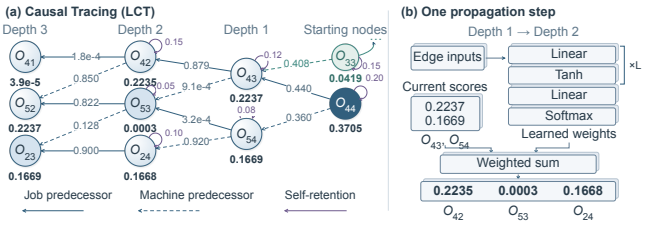


Figure 3: Causal tracing and probability propagation along selected paths.

Table 1: Neighborhood operators and their applicability to different problem families.

Granularity	Operator	JSP/FSP	FJSP/HFSP
Single	Route	–	✓
Single	Seq-Swap, Seq-Insert	✓	✓
Pair	Route+Route	–	✓
Pair	Route+Seq-Swap/Insert	–	✓
Pair	Seq-Swap+Seq-Swap/Insert	✓	✓

Root scoring. The contextualized candidate representation $\tilde{\mathbf{h}}_u$ is mapped by the root scoring projection $\mathrm{MLP}_{\mathrm{root}}$ into the prior retention weight w_u, the root value $V_\theta(u)$, and a bounded residual $\Delta_\theta(u)$, which together with the propagation score from the causal tracing branch determine the final score:

$$z_u = w_u z_u^{\mathrm{prior}} + V_\theta(u) + \Delta_\theta(u) + \lambda_{\mathrm{tr}} \left[\log p_{\mathrm{tr}}(u) - \frac{1}{|\mathcal{U}_t|} \sum_{w \in \mathcal{U}_t} \log p_{\mathrm{tr}}(w) \right]. \tag{3}$$

Here, λ_{tr} controls the contribution of causal tracing. After invalid candidates are masked, at most K operations are sampled without replacement according to z_u, forming the intervention target set U_t. The causal reasoning network, including LCT, is trained using makespan-improvement feedback.

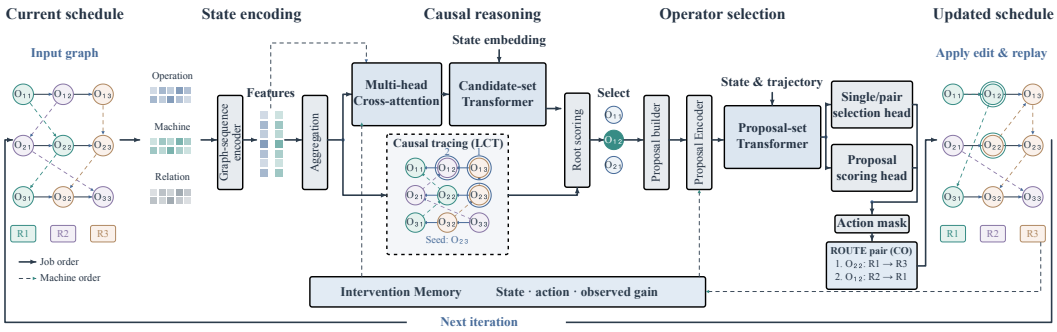


Figure 4: The network architecture of CausaSched.

4.3 Operator Selection

Around U_t, the operator policy constructs Route, Seq-Swap, Seq-Insert, and bounded two-operator compositions subject to machine eligibility and precedence constraints (Table 1); no rerouting candidates are generated for JSP or FSP.

Compositional operators (CO). The effectiveness of an atomic operator is constrained by the current machine assignment and processing order, such that a potentially beneficial modification may become ineffective when no suitable action position is available. CO allows two compatible atomic modifications to be executed jointly, enabling one operator to reshape the local schedule and create more favorable conditions for the other, thereby enlarging the reachable neighborhood beyond independent atomic moves (Appendix C, Figure 5).

Operator selection. The Proposal builder generates actions a around U_t and maps them to features $\mathbf{f}_a$. The Proposal Encoder maps these features to a latent representation $\mathbf{h}_a^0$ and aggregates contextual information to form features $\mathbf{h}_a$. Current state and candidate-pool information are encoded into state features $\mathbf{h}_s$, while trajectory information is encoded into trajectory features $\mathbf{h}_\tau$. The Proposal-set Transformer jointly processes these three types of features as $[\tilde{\mathbf{h}}_s; \tilde{\mathbf{h}}_\tau; \{\tilde{\mathbf{h}}_a\}_{a \in \mathcal{A}_t}] = \mathrm{Trans}_{\mathrm{op}}([\mathbf{h}_s; \mathbf{h}_\tau; \{\mathbf{h}_a\}_{a \in \mathcal{A}_t}])$. The representations $\mathbf{h}_a^0$ and $\tilde{\mathbf{h}}_a$ map to a base score b_a and a bounded residual, respectively; their sum gives z_a, with the associated parameters jointly trained. Meanwhile, the mean- and max-pooled candidate representations are combined with the state representation $\tilde{\mathbf{h}}_s$ and trajectory representation $\tilde{\mathbf{h}}_\tau$ and fed into the granularity-preference head $\mathrm{MLP}_{\mathrm{gran}}$, producing the preference δ_k for single and paired moves.

The candidate set is partitioned as $\mathcal{A}_t = \mathcal{A}_t^{(1)} \uplus \mathcal{A}_t^{(2)}$, where $\uplus$ denotes disjoint union and the superscripts distinguish single and paired moves. For $a \in \mathcal{A}_t^{(k)}$, $k \in \{1, 2\}$, the granularity score and final action probability are

$$g_k = \log\left(\frac{1}{|\mathcal{A}_t^{(k)}|}\sum_{a\in\mathcal{A}_t^{(k)}} e^{z_a}\right) + \delta_k,$$
$$p_\phi(a) = \mathrm{softmax}(g)_k \frac{e^{z_a}}{\sum_{b\in\mathcal{A}_t^{(k)}} e^{z_b}}. \tag{4}$$

This factorization first allocates probability mass between single and paired moves and then selects a concrete modification within the corresponding granularity. The log-mean-exp term in Eq. (4) avoids granularity bias caused by different candidate counts, while empty groups are masked out.

Multi-trajectory candidate search. Tracing associated branches and using multiple operator types gives CausaSched a broad intervention candidate space. To explore it, we unfold R trajectories from the same schedule. Each step constructs $\mathcal{A}_t$ around U_t and samples the action sequence $D_t = (a_{t,1}, \ldots, a_{t,n_t})$ without replacement, where $a_{t,j} \in \mathcal{A}_t$ and $n_t \le N_c$, with N_c the per-step candidate limit. Each action is replayed from the current schedule, and the feasible action yielding the smallest makespan is executed. Trajectories may temporarily worsen makespan, while subsequent rounds start from the retained best schedule (Appendix Figure 6). The operator policy is likewise trained using makespan-improvement feedback.

4.4 Causal Hierarchical Group Relative Policy Optimization

CH-GRPO combines group-relative advantages (Shao et al., 2024) with PPO clipping (Schulman et al., 2017) to train target selection and action sampling jointly. The policy factorizes as $\pi_{\theta,\phi}(U_t, D_t \mid s_t) = \pi_\theta^{\text{root}}(U_t \mid s_t)\pi_\phi^{\text{op}}(D_t \mid s_t, U_t, \mathcal{A}_t)$.

Step reward and advantage. Using $r_{i,t}$ from Section 4.1, each policy $x \in \{\text{root}, \text{op}\}$ computes $A_{i,t}^{x,\text{step}} = (q_{i,t}^x - \mu_t^x)/(\sigma_t^x + \epsilon_{\text{adv}})$. Here, q^x is the immediate reward or its mean over matching decisions (grouping in Appendix D).

Trajectory reward and advantage. r_i^{traj} measures the best improvement from the shared start. Normalizing over R trajectories with the same budget gives $A_i^{\text{traj}} = (r_i^{\text{traj}} - \mu_{\text{traj}})/(\sigma_{\text{traj}} + \epsilon_{\text{adv}})$. We combine step and trajectory advantages as $\widetilde{A}_{i,t}^x = (1-\eta)A_{i,t}^{x,\text{step}} + \eta\mathbb{I}[t < \tau_i^\star]A_i^{\text{traj}}$. Weight $\eta \in [0, 1]$ controls trajectory feedback, assigned only before the first attainment of $C_i^\star$ at $\tau_i^\star$; step feedback covers all steps.

Joint training. Let $\rho_{i,t}^x = \pi_x/\pi_x^{\text{old}}$ be the current-to-behavior probability ratio for the same sampled decision. The PPO clipped objective is

$$\mathcal{S}_x = \mathbb{E}_{i,t}\left[\min\left(\rho_{i,t}^x \widetilde{A}_{i,t}^x, \mathrm{clip}(\rho_{i,t}^x, 1-\epsilon, 1+\epsilon)\widetilde{A}_{i,t}^x\right)\right]. \tag{5}$$

We minimize $\mathcal{L} = -\mathcal{S}_{\text{op}} - \lambda_{\text{root}}\mathcal{S}_{\text{root}}$, with $\lambda_{\text{root}} = 1$ and clipping threshold ϵ. Root terms average conditional draws; operator terms use the joint probability of D_t, not the winner's single-draw probability. Appendix Algorithm 1 gives the procedure.

5 Experiments

We evaluate CausaSched on public FJSP benchmarks, public benchmarks of other scheduling types, and mixed synthetic instances. Supplementary data and search configurations appear in Appendices A and D.

5.1 Experimental Setup

Datasets. We train a unified policy on 128 public and synthetic instances, comprising 22 JSP, 21 FSP, 64 FJSP, and 21 HFSP instances, with 4–20 jobs and 4–20 processing resources. Evaluation

Table 2: Performance on public FJSP benchmarks by problem size.

	Brandimarte								Hurink Rdata								Behnke–Geiger					
	10 × 10		20 × 10		20 × 15		30 × 15		10 × 10		15 × 10		20 × 10		30 × 10		10 × 20		20 × 20		50 × 20	
Method	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
CP-SAT	0.00%	0.1h	0.14%	6.2s	2.03%	1h	1.76%	2.5m	0.00%	9.1s	0.00%	0.4h	0.31%	1h	1.85%	9.3m	0.00%	1.2s	0.00%	8.1m	0.00%	1h
DANIEL (S)	28.07%	2.7s	1.95%	5.2s	11.68%	6.8s	4.13%	9.4s	8.60%	1.3s	14.48%	2.5s	11.08%	4.0s	4.99%	8.3s	13.51%	0.7s	23.28%	2.1s	17.61%	14.2s
HGNN (S-10)	64.91%	4.3s	5.37%	15.7s	36.04%	16.1s	16.82%	19.1s	13.46%	1.9s	16.99%	4.3s	12.15%	8.4s	6.37%	26.3s	48.15%	0.5s	49.00%	2.0s	44.14%	20.1s
DANIEL (G)	35.09%	1.2s	3.91%	1.3s	12.18%	1.2s	8.09%	1.5s	15.94%	0.5s	28.49%	0.7s	21.16%	1.0s	9.93%	1.5s	21.33%	0.2s	32.85%	0.5s	21.90%	1.3s
HGNN (G)	68.42%	1.0s	4.23%	1.8s	35.03%	2.1s	18.47%	2.6s	26.62%	0.4s	18.09%	0.8s	10.77%	1.3s	3.83%	2.9s	53.11%	0.2s	66.39%	0.4s	53.18%	2.2s
GD-100	18.74%	2.6m	1.73%	4.7m	7.42%	5.9m	4.86%	7.3m	5.82%	51.8s	10.91%	1.7m	6.54%	3.1m	3.58%	6.5m	14.22%	30.7s	17.48%	1.8m	13.74%	12.6m
FI-100	22.16%	3.2m	1.59%	5.4m	6.18%	7.1m	4.02%	8.5m	7.11%	58.8s	9.47%	2.3m	8.23%	3.9m	3.31%	7.8m	11.87%	39.7s	21.26%	2.5m	12.62%	15.4m
BI-100	20.03%	3.1m	1.47%	5.7m	6.83%	6.6m	3.66%	8.9m	6.36%	56.4s	8.61%	2.1m	7.11%	4.2m	2.94%	8.3m	13.05%	36.2s	19.03%	2.7m	11.44%	14.8m
CausaSched-100	**17.82%**	**14.6s**	**1.45%**	**24.8s**	**5.42%**	**22.7s**	**2.18%**	**46.3s**	**5.41%**	**5.8s**	**8.92%**	**13.1s**	**5.52%**	**26.2s**	**2.73%**	**42.4s**	**10.74%**	**4.8s**	**12.83%**	**18.6s**	**10.26%**	**1.6m**
GD-500	18.74%	13.5m	1.73%	25.1m	7.42%	30.1m	3.94%	39.2m	5.82%	4.5m	10.91%	8.8m	6.54%	15.9m	3.58%	33.3m	14.22%	2.7m	16.62%	9.4m	13.28%	64.6m
FI-500	22.16%	16.7m	1.31%	27.6m	6.18%	37.2m	4.02%	43.1m	7.11%	5.1m	9.47%	12.1m	8.23%	20.3m	3.31%	41.4m	11.87%	3.5m	20.31%	12.9m	12.31%	78.2m
BI-500	20.03%	15.4m	1.47%	30.2m	6.83%	33.6m	3.08%	46.1m	6.36%	4.9m	8.61%	10.8m	7.11%	21.9m	2.52%	41.9m	13.05%	3.2m	18.12%	13.8m	11.17%	78.3m
CausaSched-500	**16.87%**	**1.5m**	**1.32%**	**2.2m**	**4.87%**	**1.7m**	**1.48%**	**3.9m**	**5.26%**	**48.5s**	**7.57%**	**51.0s**	**5.27%**	**2.3m**	**2.18%**	**3.6m**	**7.46%**	**24.5s**	**9.27%**	**1.8m**	**8.82%**	**8.1m**

References: published best-known values for Brandimarte and Hurink Rdata, updated by our CP-SAT runs; CP-SAT solutions for Behnke–Geiger, where published references are unavailable. Time units: s, m (minutes), h (hours). S/G: sampled/greedy decoding; DANIEL uses 100 samples and HGNN 10. Iterative methods are evaluated under inference budgets of 100 and 500 steps.

covers public FJSP benchmarks, public benchmarks of other scheduling types, and the Mixed-Synth dataset, with instance sizes reported in Tables 2 and 3.

Baselines. On FJSP benchmarks, we compare against the DRL-based construction methods DANIEL and HGNN, the improvement heuristics GD, FI, and BI, and CP-SAT (Perron et al., 2023). For other scheduling types, we additionally include the applicable methods L2D, L2S, NEH (Nawaz et al., 1983), and an IG-based implementation (Ruiz & Stützle, 2007). Each method is evaluated only on the problem types it supports.

Metrics and budgets. Makespan (Obj.) and runtime (Time) are arithmetic means across instances; relative gaps (Gap) and percentage reductions from initial schedules are computed per instance before averaging. Specifically, $\text{Gap}_i = (C_i - C_i^{\text{ref}})/C_i^{\text{ref}} \times 100\%$, where C_i^{ref} is the reference makespan for instance i, with sources given in the table notes.

5.2 Public FJSP Benchmarks

We evaluate CausaSched on the public Brandimarte (Brandimarte, 1993), Hurink Rdata (Hurink et al., 1994), and Behnke–Geiger (Behnke & Geiger, 2012) benchmarks, with results reported in Table 2. DANIEL and HGNN use the official implementations and pretrained weights released by their authors (Wang et al., 2024; Song et al., 2023). Under inference budgets of 100 and 500 steps, CausaSched achieves better solution quality than DRL-based construction methods and rule-based improvement heuristics in most size groups. Increasing the budget from 100 to 500 steps further reduces Gap in every size group.

5.3 Other Public Scheduling Benchmarks

To assess cross-family applicability, we evaluate CausaSched on the public Lawrence (JSP) (Lawrence, 1984), Taillard (FSP) (Taillard, 1993), and Carlier–Néron (HFSP) (Carlier & Néron, 2000) benchmarks. L2D and L2S use their official implementations (Zhang et al., 2020; 2024); IG (NP) denotes non-permutation iterated greedy. The test instances span 10–20 jobs and 5–29 machines. As shown in Table 3, under a 500-step budget, CausaSched achieves a Gap of 0.0% on JSP, matching L2S, and Gaps of 2.18% and 12.16% on FSP and HFSP, respectively, outperforming the corresponding baselines in the table. Increasing the budget from 100 to 500 steps improves solution quality in all three families, supporting the method's applicability across scheduling types.

5.4 Mixed-Instance Evaluation

Mixed-Synth contains twenty-four synthetic test instances each of JSP, FSP, FJSP, and HFSP, evaluated using the same policy without family-specific fine-tuning; Gap uses CP-SAT-proven optima as references. As shown in Table 3, CausaSched achieves mean makespans of 644.5 and 632.2 under budgets of 100 and 500 steps, respectively, outperforming GD, FI, and BI at both budgets. The Gap,

Table 3: Cross-family benchmarks and a mixed synthetic set.

	Lawrence (JSP)			Taillard (FSP)			C–N (HFSP)			Mixed-Synth		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
L2D	987.0	15.79%	0.16s	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
L2S-100	870.6	1.30%	2.8s	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
NEH	N/A	N/A	N/A	1374.8	3.21%	0.13s	N/A	N/A	N/A	N/A	N/A	N/A
IG (NP)-100	N/A	N/A	N/A	1431.6	5.13%	1.3s	N/A	N/A	N/A	N/A	N/A	N/A
GD-100	882.4	2.69%	52.3s	1382.4	3.37%	2.3m	122.7	19.37%	1.4m	651.4	12.71%	4.7m
FI-100	871.4	1.40%	59.1s	1447.2	5.63%	2.8m	120.2	16.93%	1.9m	657.8	14.05%	5.4m
BI-100	871.6	1.42%	56.8s	1458.7	5.99%	2.5m	126.3	21.05%	1.6m	664.3	14.62%	5.1m
CausaSched-100	**871.4**	**1.40%**	**6.2s**	**1385.3**	**4.13%**	**11.1s**	**117.6**	**14.36%**	**7.8s**	**644.5**	**11.37%**	**21.5s**
L2S-500	859.3	0.00%	12.2s	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
IG (NP)-500	N/A	N/A	N/A	1431.6	5.13%	5.8s	N/A	N/A	N/A	N/A	N/A	N/A
GD-500	882.4	2.69%	4.6m	1382.4	3.37%	12.2m	122.7	19.37%	7.1m	651.4	12.71%	24.3m
FI-500	871.4	1.40%	5.4m	1447.2	5.63%	14.8m	120.2	16.93%	9.8m	657.8	14.05%	28.7m
BI-500	871.6	1.42%	5.0m	1458.7	5.99%	13.5m	121.1	18.42%	8.5m	664.3	14.62%	26.4m
CausaSched-500	**859.3**	**0.00%**	**31.5s**	**1358.7**	**2.18%**	**56.3s**	**115.3**	**12.16%**	**39.5s**	**632.2**	**9.24%**	**1.8m**

References: best-known makespans for Lawrence, published bounds for Taillard, and proven optima for C–N. Time is in seconds. Iterative methods are evaluated under inference budgets of 100 and 500 steps.

equally weighted across the four families, decreases from 11.37% to 9.24%. These results support the effectiveness of the unified policy on mixed scheduling instances.

In our tests, larger or more constrained instances, especially FJSP with multiple eligible machines, increase the time costs of solution construction and feasibility checks.

5.5 Component Ablations and Candidate-Count Sensitivity

We assess component contributions using a shared checkpoint and 200-step budget on 64 instances, with full results and search curves in Appendix E. Full achieves the highest overall reduction of 13.94%, compared with 12.41% for Uniform-LCT, supporting learned propagation for prioritizing effective intervention targets. Removing CO reduces improvement to 12.82%, consistent with the value of coordinating modifications beyond single-operator neighborhoods. Without Memory, improvement falls to 13.05%, indicating that historical intervention information benefits action selection. Disabling cross-attention or GAT causes the largest drops, to 11.73% and 11.84%, respectively, highlighting the importance of target–factor context and structural dependencies. Together, these comparisons support the complementary roles of intervention targeting, neighborhood construction, and contextual information in the current policy's overall performance, while family-specific gains vary. On the same instances under 100- and 500-step budgets, mean makespan first decreases and then increases as the candidate count N_c grows (Appendix Table 8).

6 Conclusion and Limitations

We propose CausaSched for multiple scheduling types, combining makespan-oriented causal tracing with compound-operator selection and jointly learning both intervention policies through CH-GRPO. Experiments on public benchmarks and mixed instances support its schedule-improvement capability, while component ablations demonstrate contributions to overall performance. The present study focuses on makespan minimization under static, deterministic conditions; dynamic job arrivals, machine failures, and uncertain processing times remain unevaluated. Future work will extend the framework to dynamic and uncertain scheduling and consider multiple objectives.

AI use statement

We used generative AI for language polishing, literature discovery, and LLM-assisted generation of training examples during training-data construction (Appendix B.4). No generative AI was used for other research tasks requiring disclosure. We reviewed all AI-assisted outputs and take full responsibility for the final manuscript and results.

References

Dirk Behnke and Martin Josef Geiger. Test instances for the flexible job shop scheduling problem with work centers. Technical Report RR-12-01-01, Helmut-Schmidt University, 2012. URL https://openhsu.ub.hsu-hh.de/entities/publication/436.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. doi: 10.1016/j.ejor.2020.07.063.

Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993. doi: 10.1007/BF02023073.

Kailash Budhathoki, Dominik Janzing, Patrick Bloebaum, and Hoiyi Ng. Why did the distribution change? In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 1666–1674, 2021. URL https://proceedings.mlr.press/v130/budhathoki21a.html.

Kailash Budhathoki, Lenon Minorics, Patrick Bloebaum, and Dominik Janzing. Causal structure-based root cause analysis of outliers. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2357–2369, 2022. URL https://proceedings.mlr.press/v162/budhathoki22a.html.

Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. doi: 10.1057/jors.2013.71.

Jacques Carlier and Emmanuel Néron. An exact method for solving the multi-processor flow-shop. *RAIRO. Operations Research*, 34(1):1–25, 2000. URL https://numdam.org/item/RO_2000__34_1_1_0/.

Chun-Lung Chen and Chuen-Lung Chen. A bottleneck-based heuristic for minimizing makespan in a flexible flow line with unrelated parallel machines. *Computers & Operations Research*, 36(11): 3073–3081, 2009. doi: 10.1016/j.cor.2009.02.004. URL https://doi.org/10.1016/j.cor.2009.02.004.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/131f383b434fdf48079bff1e44e2d9a5-Abstract.html.

Yuanzhu Di, Libao Deng, and Lili Zhang. A collaborative-learning multi-agent reinforcement learning method for distributed hybrid flow shop scheduling problem. *Swarm and Evolutionary Computation*, 91:101764, 2024. doi: 10.1016/j.swevo.2024.101764.

Jonas K. Falkner, Daniela Thyssens, Ahmad Bdeir, and Lars Schmidt-Thieme. Learning to control local search for combinatorial optimization. In *Machine Learning and Knowledge Discovery in Databases: ECML PKDD 2022*, pp. 361–376, 2022. doi: 10.1007/978-3-031-26419-1_22.

Xiao Han, Saima Absar, Lu Zhang, and Shuhan Yuan. Root cause analysis of anomalies in multivariate time series through Granger causal discovery. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/6fde96479648d71e4fd9724374bf76eb-Paper-Conference.pdf.

Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15(4):205–215, 1994. doi: 10.1007/BF01719451.

James E. Kelley and Morgan R. Walker. Critical-path planning and scheduling. In *Proceedings of the Eastern Joint IRE-AIEE-ACM Computer Conference*, pp. 160–173, 1959. doi: 10.1145/1460299.1460318.

S. Lawrence. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

Kathrin Maassen, Paz Perez-Gonzalez, and Lisa C. Günther. Relationship between common objective functions, idle time and waiting time in permutation flow shop scheduling. *Computers & Operations Research*, 121:104965, 2020. doi: 10.1016/j.cor.2020.104965. URL https://doi.org/10.1016/j.cor.2020.104965.

Muhammad Nawaz, E. Emory Enscore, Jr., and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983. doi: 10.1016/0305-0483(83)90088-9.

Fei Ni, Jianye Hao, Jiawen Lu, Xialiang Tong, Mingxuan Yuan, Jiahui Duan, Yi Ma, and Kun He. A multi-graph attributed reinforcement learning based optimization algorithm for large-scale hybrid flow shop scheduling problem. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 3441–3451. Association for Computing Machinery, 2021. doi: 10.1145/3447548.3467135. URL https://doi.org/10.1145/3447548.3467135.

Junyoung Park, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park. Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, 59(11):3360–3377, 2021. doi: 10.1080/00207543.2020.1870013.

Laurent Perron, Frédéric Didier, and Steven Gay. The CP-SAT-LP solver. In *29th International Conference on Principles and Practice of Constraint Programming*, volume 280 of *Leibniz International Proceedings in Informatics*, pp. 3:1–3:2, 2023. doi: 10.4230/LIPIcs.CP.2023.3.

Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007. doi: 10.1016/j.ejor.2005.12.009.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19 (2):1600–1610, 2023. doi: 10.1109/TII.2022.3189725.

E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. doi: 10.1016/0377-2217(93)90182-M.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Jiaqi Wang, Zhiguang Cao, Peng Zhao, Rui Cao, Yubin Xiao, Yuan Jiang, and You Zhou. Learning memory-enhanced improvement heuristics for flexible job shop scheduling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 82849–82877, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/77629f2df5ef97ad967157c69cd8af7b-Paper-Conference.pdf.

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

Shengluo Yang, Zhigang Xu, and Junyi Wang. Intelligent decision-making of scheduling for dynamic permutation flowshop via deep reinforcement learning. *Sensors*, 21(3):1019, 2021. doi: 10.3390/s21031019.

ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, 2007. doi: 10.1016/j.cor.2005.12.002.

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 *Advances in Neural Information Processing Systems*, volume 33, pp. 1621–1632, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/11958dfee29b6709f48a9ba0387a2431-Paper.pdf.

Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/c75abb33341363ee874a71f81dc45a3a-Paper-Conference.pdf.

Appendix

A Data and Experimental Settings

A.1 Data Composition

The training set contains 128 public and synthetic instances, with family counts and size ranges in Table 4. Initial schedules use the earliest-finish rule: among precedence-ready operations and their eligible machines, the rule selects the assignment with the earliest projected completion. Public evaluation uses Brandimarte, Hurink Rdata, Behnke–Geiger, Lawrence, Taillard, and Carlier–Néron instances; size groups and comparisons appear in Tables 2 and 3. Mixed-Synth serves the mixed-family main experiment. Component ablations use 64 medium-sized instances from a separate synthetic set, with sixteen per family.

Table 4: Training and separate ablation sets. J denotes jobs and M total machines; ranges are reported per family.

	Training			Ablation	
Family	Count	J	M	Count	J / M
JSP	22	5–16	4–10	16	10–15 / 5–10
FSP	21	5–16	4–10	16	10–15 / 5–10
FJSP	64	4–20	4–20	16	10–15 / 5–10
HFSP	21	5–16	6–18	16	10–15 / 11–13
Total	128			64	

A.2 Synthetic Instance Generation

The ablation instances use $(J, m, p_{\max}) = (10\text{–}15, 5\text{–}10, 100)$, with sixteen instances per family and processing times sampled uniformly from $[1, p_{\max}]$. JSP independently permutes the m machines for each job, whereas FSP shares their visiting order. FJSP has $\max(3, \lfloor m/2 \rfloor)$ operations per job and at most three eligible machines per operation. HFSP uses $\max(3, \lfloor m/2 \rfloor)$ stages with two to four parallel machines each; here m is a generation parameter, not the total HFSP machine count.

B Additional Method Details

B.1 Critical Paths and Intervention Tracing

With machine assignments and processing orders fixed, operation v can start only when both its job and machine are ready: $s_v = \max(r_v^{\text{job}}, r_v^{\text{mac}})$, where the two terms are the completion times of its job and machine predecessors. For an earliest-start schedule, trace backward from an operation attaining makespan through predecessors satisfying $c_u = s_v$. This yields a chain of operations that meet end to start and extends to the schedule origin. Its total processing time equals makespan, defining a critical path (Kelley & Walker, 1959; Zhang et al., 2007); tied tight constraints can yield multiple critical paths.

Delay entry points identify temporal mismatches in the current schedule: $r_v^{\text{mac}} > r_v^{\text{job}}$ means that the job waits for the machine, while the reverse means that the machine waits for the job. Equal readiness times yield no such gap and therefore do not define a waiting- or idle-type delay entry point. Concentrated processing loads provide an additional resource-occupancy signal. We focus on delay blocks whose effects reach makespan through tight dependency chains. Such chains form segments of critical paths; not every waiting or heavily loaded operation is critical.

Starting from these blocks, tracing follows job and machine predecessors to seek upstream sources layer by layer. An operation may wait because its predecessor finishes late, while that predecessor may itself be delayed by upstream resource competition or job readiness. Responsibility is therefore traced beyond immediate neighbors. Considering both tight and other associated predecessor relations also reaches branches outside the current critical path, where assignment or order changes may

alter resource availability for critical operations. Operations not selected as delay entry points can still become upstream intervention targets.

The weights $T_\theta^{(\ell)}$ in Section 4.2 and Appendix B.3 allocate responsibility at each layer, accumulating contributions when different delay sources reach the same operation. The causal reasoning network is trained using realized makespan improvements, aligning tracing preferences with the scheduling objective. If an intervention alleviates a local issue but introduces new waiting or resource conflicts, its net effect on makespan still contributes to the learning feedback.

B.2 State Encoding and Intervention Memory

State encoding comprises relational graph encoding, sequence encoding and fusion, and reverse-dependency encoding. Graph and sequence features are extracted in parallel, fused, and passed to the reverse-dependency encoder. Graph-sequence encoder in Figure 4 covers relational graph encoding and sequence encoding with fusion. The heterogeneous scheduling graph contains operation, job, machine, and processing-mode nodes. Relations include job precedence and same-machine order between operations, machine eligibility and current assignment between operations and machines, and links to jobs and processing modes. In relational graph encoding, type-specific projections map node attributes into a common hidden space. A relation-aware graph attention network (GAT) separately encodes context/feasibility relations and scheduling dependencies, incorporating edge types and edge features; the two graph streams are averaged. Second, a Transformer encodes the supplied Gantt features for each schedule. These features describe processing arrangements, rather than operator candidates. Feature representations are aligned to operation nodes by their identifiers, averaged when multiple features map to one node, and fused with the graph representations through a learned gate and layer normalization (Veličković et al., 2018; Vaswani et al., 2017).

The graph encoder is summarized as

$$H^G = \mathrm{GAT}_{\theta_G}(G_t, X_t, E_t), \qquad H^G = \{\mathbf{h}_u^G\}_u. \tag{6}$$

Here G_t is the heterogeneous scheduling graph, X_t contains node features, and E_t contains edge types and features. GAT_{θ_G} includes the two relational graph streams and their mean aggregation. With aligned sequence representations $\mathbf{h}_u^S$, fusion is

$$\mathbf{h}_u^F = \mathrm{LN}\big(\mathbf{g}_u \odot \mathbf{h}_u^G + (1-\mathbf{g}_u) \odot \big[\mathbf{h}_u^G + P_S(\mathbf{h}_u^S)\big]\big)\,, \tag{7}$$

where P_S is a linear projection and $\mathbf{g}_u$ is a learned gate conditioned on graph and sequence context. Third, the fused node representations initialize a gated residual encoder over reversed job-precedence and machine-order edges. Its outputs represent candidate intervention targets, while pooling nodes within makespan-related blocks forms factor queries. Target–factor cross-attention corresponds to Multi-head Cross-attention in the Causal reasoning region of Figure 4; its outputs feed the Candidate-set Transformer and Root scoring. The operator branch in the Operator selection region combines affected-operation representations, modification features, and retrieval evidence through the Proposal Encoder; the Proposal-set Transformer then compares current candidate actions.

The target and operator branches project concatenated features $\mathbf{x}$ into feature vectors using

$$\phi(\mathbf{x}) = \mathrm{GELU}(\mathrm{LN}(W\mathbf{x} + \mathbf{b}))\,. \tag{8}$$

Feature types use separate projection parameters. Both candidate-set Transformers use Pre-LayerNorm. For an input feature matrix H, one layer computes

$$\widehat{H} = H + \mathrm{MHA}(\mathrm{LN}_1(H))\,, \qquad H' = \widehat{H} + \mathrm{FFN}\Big(\mathrm{LN}_2(\widehat{H})\Big)\,. \tag{9}$$

Here MHA is multi-head self-attention with validity masks, and FFN is a feature-wise feed-forward network. LN standardizes each feature across its dimensions with learned scale and bias, independently of other instances in the batch.

Intervention Memory, shown below the networks in Figure 4, stores state features, intervention types, and observed execution gains, supplying retrieved experience to both target and operator selection. Retrieval uses state proximity to produce evidence such as support counts, success rates, and gains, which informs the policy without imposing actions. Trajectory-local records follow the retained state prefix and contain only interventions executed before the current decision. Test records do not enter the training retrieval store.

B.3 LCT Transition Weights and Expansion Rules

This section details Causal reasoning in Figure 2, specifically the Causal tracing branch within Causal reasoning in Figure 4. Figure 3 illustrates its reverse propagation paths and inter-layer scoring network. Let $\mathbf{x}_{uv}^{(\ell)}$ concatenate node-pair representations and search context at tracing layer ℓ. The propagation scorer stacks L Linear–Tanh blocks, followed by a linear output layer producing a scalar score: Initialize $\mathbf{h}_{uv,0}^{(\ell)} = \mathbf{x}_{uv}^{(\ell)}$. Hidden representations are updated as

$$\mathbf{h}_{uv,j}^{(\ell)} = \tanh\left(W_j \mathbf{h}_{uv,j-1}^{(\ell)} + \mathbf{b}_j\right), \qquad j = 1, \ldots, L. \tag{10}$$

The final propagation score is

$$e_\theta^{(\ell)}(u, v) = \mathbf{w}_{\text{out}}^\top \mathbf{h}_{uv,L}^{(\ell)} + b_{\text{out}}. \tag{11}$$

Normalizing these scores over the node itself and its predecessors gives the transition weights in Section 4.2: $T_\theta^{(\ell)}(u|v) = \text{softmax}_{u \in \{v\} \cup \text{Pa}(v)} e_\theta^{(\ell)}(u, v)$. As in the main text, $\text{Pa}(v)$ denotes job and machine predecessors, and $u = v$ corresponds to self-retention; invalid or unretained tracing edges are masked. State dependence is included in $\mathbf{x}_{uv}^{(\ell)}$. These weights propagate $q^{(\ell)}$, whose layers are mixed by β_ℓ to obtain $p_{\text{tr}}(u)$ for root scoring. L denotes the number of repeated Linear–Tanh blocks, whereas D denotes the tracing depth reached at the current decision. Each source retains at most four upstream operations per layer: three by propagation score and one for exploration. This width limit applies to the source's entire layer, not to each node. Expansion follows legal predecessors layer by layer, with contributions from different sources accumulating at shared operations. Representation and scoring parameters are shared across layers, with mixing normalized over the expanded layers $0, \ldots, D$ rather than a fixed-length depth output.

B.4 Supervised Initialization

Supervised initialization provides initial parameters for the scoring and policy components in Figure 4 and is separate from online scheduling search; the LLM participates only in offline data preparation, not as an online module in Figure 2. Offline data preparation includes LLM-assisted construction of intervention experience. For a current schedule from a training instance, the LLM receives a state summary, candidate tracing chains, intervention targets, allowed operators, and a list of legal edits. It proposes targets and concrete modifications within this scope, returning structured edits and supporting reasons. The LLM thus helps propose search candidates rather than generating makespan or reward labels.

Candidates first undergo format and legality checks, then the system executes them while keeping existing decisions outside the local modification scope fixed. The system checks that the requested edits are executed and the resulting schedule is feasible, and computes the makespan change from the actual schedule. Only verified state–intervention–outcome records enter the experience store. LLM-generated reasons and confidence are retained as auxiliary metadata, not used to compute rewards or determine acceptance.

Supervision comes from actual execution and schedule replay; candidate sources also include legal neighborhood enumeration. The corresponding supervision initializes scoring and policy components, followed by reinforcement learning using rewards from actual schedule transitions. Online training and inference require no LLM calls.

Supervised operator selection. For a training state S, let $\mathcal{A}_S^{\text{sup}}$ be its candidate pool, $U_S(a)$ the execution-derived utility, and $s_\psi(S, a)$ the selector logit. The supervised label a_S^* is the highest-utility feasible candidate with positive utility; if none exists, it is STOP, whose utility is zero. The selector distribution and classification loss are

$$p_\psi(a \mid S) = \frac{\exp s_\psi(S, a)}{\sum_{b \in \mathcal{A}_S^{\text{sup}} \cup \{\text{STOP}\}} \exp s_\psi(S, b)}, \qquad \mathcal{L}_{\text{top1}}(S) = -\log p_\psi(a_S^* \mid S). \tag{12}$$

To preserve relative utility information beyond the best label, sampled pairs $\mathcal{P}_S$ satisfy $U_S(a) > U_S(b)$. The initialization objective is

$$\mathcal{L}_{\text{SFT}} = \mathbb{E}_S \left[\mathcal{L}_{\text{top1}}(S) + \frac{\eta}{|\mathcal{P}_S|} \sum_{(a,b) \in \mathcal{P}_S} w_{ab} \max\{0, 1 - s_\psi(S, a) + s_\psi(S, b)\} \right]. \tag{13}$$

Here $\eta = 0.3$ and $w_{ab} = \text{clip}(|U_S(a) - U_S(b)|/20, 0.2, 5)$; the pair term is zero for an empty pair set. At most 64 pairs are sampled per state. These equations describe the retained supervised operator selector, not the subsequent joint RL objective. In particular, the supervised STOP label does not require online search to reject temporarily worsening moves. LLM text and confidence do not appear in either loss.

C Operators and Compound-Move Illustration

C.1 Compound Construction Details

Operator types and applicability are listed in Table 1. Companion selection uses operation and machine relationships rather than enumerating the full Cartesian product of atomic moves. Candidates must satisfy machine eligibility, precedence, and resource non-overlap constraints.

C.2 Controlled Illustration of Compound Moves

Figure 5 illustrates how compound operators work through complementary modifications. In the initial schedule, M3, which processes O_{41}, carries a high processing-time load: the operations assigned to it cumulatively occupy substantial processing time. O_{41} completes after the preceding operations on M3, and the effect of this time occupancy propagates through its successors O_{42} and O_{43} to the final makespan. Rerouting O_{41} alone from M3 to M2 (move B) delays its processing behind O_{32}, increasing makespan from 62 to 75. The compound move first reroutes O_{32} from M2 to M4 (move A), freeing a processing position on M2 for O_{41}, and then moves O_{41} into that position. Its start time advances from 45 to 20, reducing makespan to 50.

Move A alone leaves makespan at 62, yet enables move B to improve the schedule. This enabling role is not captured by immediate improvement, so move A may be overlooked by search that prioritizes single-step gains. Joint evaluation makes the benefit of freeing and using the processing position directly visible as a makespan reduction, reducing the risk of overlooking complementary moves during sequential search.

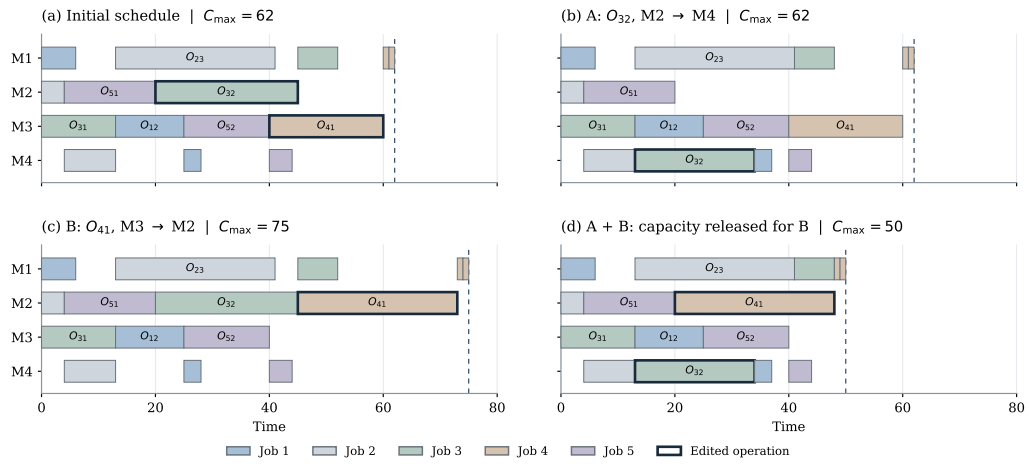


Figure 5: Same-start comparison of compound moves. A releases machine space that B uses to start earlier. Thick borders mark the relevant operations.

D Training and Search Configuration

D.1 Collection and Updates

Each training episode starts from rule-generated schedules and alternates collection and updates within a 200-step budget. Each segment expands R trajectories from a common schedule for 10

or 20 steps, comparing candidate actions as in Section 4.3 and using bounded same-state retries when blocked. The best schedule at the end of a segment initializes the next segment; a new training episode restarts from the rule-generated schedule.

Step feedback covers every executed action. At the first step of a segment, $q_{i,0}^{\text{op}} = r_{i,0}$, while the root signal averages immediate rewards over identical target-draw sequences and is standardized across distinct sequences. At subsequent steps, active trajectories collected from the same start are grouped by local index t, with $q_{i,t}^{\text{root}} = q_{i,t}^{\text{op}} = r_{i,t}$. Step advantages follow Section 4.4. Since trajectory states may differ, these comparisons provide relative feedback rather than a same-state value baseline.

Trajectory feedback compares the best makespan improvements among segment trajectories with the same start and budget, and is assigned only to actions preceding the first attainment of each trajectory's best schedule. It is combined with step advantages as defined in the main text to jointly update both policies. Probability ratios use the target and candidate-action sequences recorded during collection; the operator term corresponds to the complete candidate-draw sequence D_t, not the single-draw probability of the executed action.

Table 5: Default training and search configuration. Inference-ablation budgets are listed separately.

Category	Parameter	Value
Architecture	LCT hidden blocks L (excluding output layer)	1
	LCT scoring hidden dimension	64
	Hidden dimension of both Transformers	128
	Feed-forward dimension of both Transformers	256
	Attention heads in both Transformers	4
	Dropout in both Transformers	0
Collection	Training instances	128
	Trajectories per instance R	16
	Candidates per step per batch N_c	40
	Search steps per episode	200
	Steps per collection segment	10/20
Policy updates	Epochs per collected batch	3
	Probability-ratio clipping ϵ	0.2
	Optimizer	AdamW
	Policy parameter-group learning rate	10^{-4}
	Encoder/base-scorer learning rate	10^{-5}
	Weight decay	0
	Gradient norm limit	1
Tracing and retries	Trace depth	Adaptive
	Width per source per layer	4
	Score/exploration slots	3/1
	Maximum candidate batches per step	2
	Maximum target attempts per step	3
Inference ablations	Instances	64
	Search steps per instance	200
	CPU worker processes	16

D.2 Search Paths with Temporary Worsening

Figure 6 shows a five-job, four-machine FSP case. Swapping O_{12} and O_{32} on M2 first raises makespan from 139 to 147; swapping O_{13} and O_{33} on M3 then lowers it to 136. Rejecting the first move solely for increasing makespan would block this subsequent improvement path. We therefore allow temporary worsening within trajectories while separately retaining the best schedule found. Verified by sequential replay, this example illustrates the search rule, not the superiority of long-term over one-step rewards or the absence of alternative improving paths.

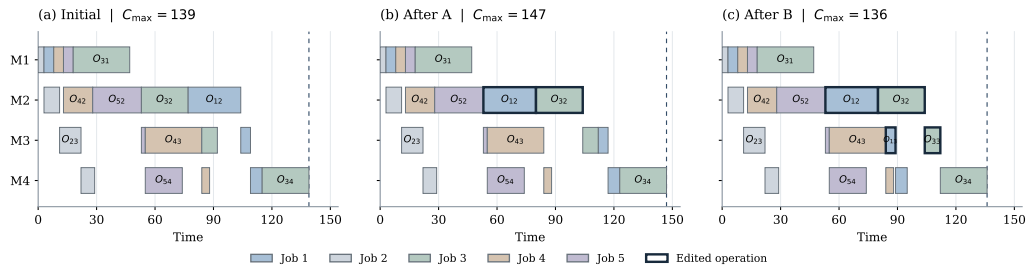


Figure 6: Improvement after temporary worsening: $139 \to 147 \to 136$. Best-so-far makespan is 139, 139, and 136, respectively.

D.3 Algorithm

Algorithm 1 summarizes training and search. H is the episode budget, h the current segment length (10/20), and E the episode count. Inference omits the advantage and parameter-update steps.

Algorithm 1 CausaSched training and search

Require: Initial schedules $\{S_j^0\}$, policies $\Theta = (\theta, \phi)$, memory $\mathcal{M}$; E, H, h, R, N_c
Ensure: Updated policies Θ and best schedules $\{S_j^\star\}$

$S_j^\star \leftarrow S_j^0 \quad \forall j$
for $e = 1, \ldots, E$ **do**
  $\bar{S}_j \leftarrow S_j^0,\ b_j \leftarrow 0 \quad \forall j$
  while $\exists j : b_j < H$ **do**
    $\mathcal{B} \leftarrow \varnothing$
    for $j : b_j < H$ **do**
      $h_j \leftarrow \min(h, H - b_j)$
      $S_{j,i,0} \leftarrow \bar{S}_j, \quad i = 1, \ldots, R$
      for each trajectory i **in parallel do**
        $\mathcal{V}_i \leftarrow \{S_{j,i,0}\}$; $\mathcal{M}_i \leftarrow \mathcal{M}$
        for $t = 0, \ldots, h_j - 1$ **do**
          Encode the current schedule and trace delay sources to obtain s_t
          Sample intervention targets $U_t \sim \pi_\theta^{\text{root}}(\cdot \mid s_t)$
          Construct single and paired target–operator actions $\mathcal{A}_t$ around U_t
          Sample up to N_c actions without replacement using π_ϕ^{op}, forming D_t
          Replay D_t from the same schedule; collect feasible successors $\mathcal{F}_t$ excluding $\mathcal{V}_i$
          if $\mathcal{F}_t = \varnothing$ **then**
            Retry untested actions or reselect targets at the same state within bounded attempts
          end if
          if $\mathcal{F}_t = \varnothing$ after retries **then break**
          Execute $S_{t+1} = \arg\min_{S \in \mathcal{F}_t} C_{\max}(S)$ and compute reward r_t
          $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, U_t, D_t, S_{t+1}, r_t, \log \pi_\Theta^{\text{old}})\}$
          $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \{S_{t+1}\}$; update trajectory-local memory $\mathcal{M}_i$
        end for
      end for
      $\bar{S}_j \leftarrow \arg\min_{S \in \{\bar{S}_j\} \cup \bigcup_i \mathcal{V}_i} C_{\max}(S)$; $S_j^\star \leftarrow \arg\min_{S \in \{S_j^\star, \bar{S}_j\}} C_{\max}(S)$
      $b_j \leftarrow b_j + h_j$
    end for
    Combine step and trajectory advantages from $\mathcal{B}$ into $\widetilde{A}_{i,t}^x$
    Jointly update Θ using the CH-GRPO objective
  end while
end for
return $\Theta, \{S_j^\star\}$

Runtime environment. All experiments ran on a single server instance with an NVIDIA RTX 4090 (24GB), an Intel Xeon Platinum 8481C CPU (25 vCPUs), and 90GB of memory, under Ubuntu 22.04, Python 3.12, PyTorch 2.8.0, and CUDA 12.8; all reported timings were measured in this environment.

E Supplementary Experiments and Component Ablations

E.1 Variant Definitions

We compare Full with five inference-time ablations on 64 synthetic instances using the same trained checkpoint. This design tests the current policy's reliance on individual components, without randomly initialized replacement networks or additional training. All variants use the same blocked-state retry rule. Table 6 defines the interventions precisely.

Table 6: Inference-time ablations. All other settings remain unchanged.

Variant	Modification
Full	Complete model.
Uniform-LCT	Uniform predecessor transitions (including self-retention) and depth mixing.
w/o CO	Single-operator candidates only.
w/o Memory	Mask historical and trajectory-local retrieval evidence.
w/o Cross-Attention	Uniform aggregation of valid factor context for each target.
w/o GAT	Disable relational graph-attention neighbor messages.

E.2 Ablation Results

The mean reduction is $\frac{1}{n}\sum_{i=1}^{n} 100(C_i^0 - C_i^{\text{best}})/C_i^0$, where C_i^0 and C_i^{best} are the initial and best makespans within budget, respectively, and $n = 64$. All ablation variants are evaluated with one candidate action sampled per step ($N_c = 1$), retaining multiple trajectories. Table 7 reports mean makespans and reductions from the initial schedules after 200 steps; Figure 7 shows the search curves.

Table 7: Inference ablation results. Makespans are arithmetic means; reduction percentages are averaged per instance. Best values in each column are bold.

Variant	JSP↓	FSP↓	FJSP↓	HFSP↓	Overall↓	Red.(%)↑
Full	**1016.50**	**1112.00**	**219.50**	422.75	**692.69**	**13.94**
Uniform-LCT	1031.75	1129.25	228.00	425.25	703.56	12.41
w/o CO	1027.75	1127.75	227.00	420.50	700.75	12.82
w/o Memory	1036.50	1120.25	222.75	423.25	700.69	13.05
w/o Cross-Attention	1037.00	1142.25	231.75	425.50	709.13	11.73
w/o GAT	1056.00	1154.75	227.75	**417.00**	713.88	11.84

Full achieves a mean makespan of 692.69 and a mean reduction of 13.94%. Uniform tracing reduces the latter to 12.41%, while disabling cross-attention and GAT yields 11.73% and 11.84%, respectively. Removing CO and Memory yields 12.82% and 13.05%. Full achieves the lowest mean makespan on JSP, FSP, and FJSP. On HFSP, w/o GAT performs slightly better than Full; this difference is influenced by the single-candidate-per-step setting used in this experiment. Figure 7 shows improvement with increasing search budget; Full attains the highest overall reduction at 200 steps. These results characterize component dependence of this checkpoint on this instance set, not performance after retraining or significance across random seeds.

On the same sixteen FJSP instances, even w/o Cross-Attention, the weakest variant on this family, achieves a mean makespan of 231.75 after 200 search steps with $N_c = 1$, below the mean of 249.50 obtained by taking the best of 100 DANIEL samples per instance, a reduction of 7.11%. Full further reduces it to 219.50.

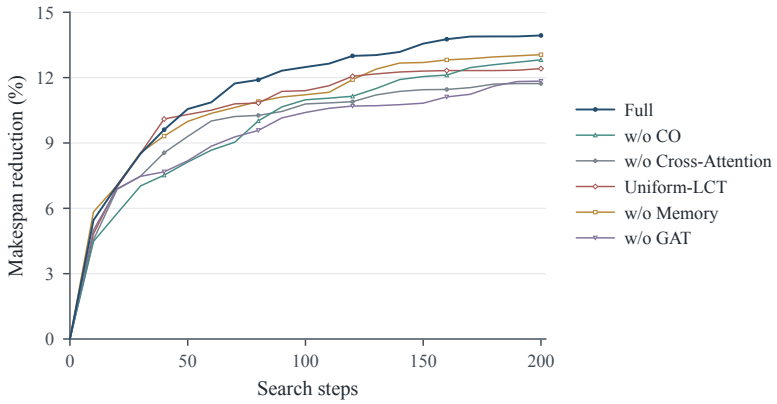


Figure 7: Search curves of six inference variants.

E.3 Candidate-Count Sensitivity

We vary the per-step candidate count N_c on the same 64 instances, fixing the checkpoint, initial schedules, seed, and $R = 16$ trajectories, under 100- and 500-step budgets. Within each budget, the comparison uses equal search depth rather than equal numbers of action evaluations.

Table 8: Candidate-count comparison at 100 and 500 steps. Makespan, percentage reduction, and time are averaged over instances.

N_c	100 steps			500 steps		
	Makespan↓	Reduction (%)↑	Time (s)↓	Makespan↓	Reduction (%)↑	Time (s)↓
20	660.94	16.85	**25.25**	633.26	19.13	**134.8**
40	658.56	17.02	29.25	627.47	19.87	156.7
60	**656.88**	**17.27**	31.38	**615.25**	**21.43**	169.3
80	659.31	17.17	31.52	619.95	20.83	172.6

Under both budgets, increasing the candidate count from 20 to 60 improves mean makespan, while 80 gives no further improvement. At 500 steps, $N_c = 60$ achieves the lowest mean makespan of 615.25 and the highest reduction of 21.43%, taking 169.3 seconds on average. All four candidate counts improve upon their respective 100-step results, indicating that longer search further improves solution quality.