SCHEDULING WITH HETEROGENEOUS RELATIONS: REINFORCEMENT LEARNING FOR COMPLEX HIERARCHICAL MANUFACTURING SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hierarchical manufacturing scheduling is particularly challenging in practice, as complex production structures and heterogeneous operational constraints give rise to tightly coupled decisions that extend substantially beyond the standard flexible job shop scheduling problem. To address this challenge, we propose **CRST**, a neural architecture that models scheduling states through multiple complementary relations. CRST performs relation-specific reasoning over workflow dependencies, machine competition, and reusable resource interactions, and then adaptively combines these complementary contexts through *Relation Mix* to measure the importance of these relations. A pair decoder directly evaluates feasible task machine assignments, while the environment enforces exact feasibility under hard scheduling constraints. Additionally, we introduce a decomposition strategy that substantially reduces memory consumption during large-scale inference while preserving global constraint consistency. Experiments show that CRST achieves solution quality comparable to CP-SAT on standard size instances and becomes more competitive on larger problems, outperforming CP-SAT under a fixed 30-minute search budget by up to 8.48% while requiring only tens of seconds of inference.

1 INTRODUCTION

Flexible job shop scheduling (FJSP) is a fundamental combinatorial optimization problem in manufacturing, and recent learning-based methods have shown that graph representations combined with deep reinforcement learning can construct competitive schedules for JSSP and FJSP (Zhang et al., 2020; Song et al., 2023; Wang et al., 2024). Closely related scheduling structures also arise beyond factories. For example, data-processing and computing clusters must schedule precedence-constrained jobs over heterogeneous processing resources while managing shared capacity (Mao et al., 2019). In the standard FJSP, each job consists of precedence-constrained operations, and each operation is assigned to one of several compatible machines. Practical manufacturing settings, however, can be substantially richer, involving multiple reusable resources, non-linear precedence structures, staged operations, and machine-availability constraints (Kasapidis et al., 2023; Perrachon et al., 2025; Gao et al., 2006). The setting studied in this work combines these challenges in a hierarchical process–stage–task structure, where tasks may require reusable resources, depend on tasks in other processes, and execute on machines with explicit rest calendars.

The main difficulty is therefore not only the number of constraints, but their *heterogeneous interaction structure*. Workflow dependencies determine how delays propagate through the production hierarchy; Task–Machine relations capture processing alternatives and machine contention; and Task–Resource relations characterize competition for reusable capacity. Existing graph-based scheduling methods largely organize decisions around operation–machine interactions (Song et al., 2023; Wang et al., 2024; Xiao et al., 2026), whereas the problem considered here requires several qualitatively different relation types to be modeled simultaneously. Mixing these interactions too early in a single homogeneous representation can obscure their distinct scheduling roles, motivating a representation that preserves relation-specific information before adaptive composition.

We therefore propose **CRST**, a constraint-relational scheduling transformer that mirrors this structure in the policy architecture. CRST separately reasons over workflow, machine, and resource rela-

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.

tions and then combines the resulting contexts through **Relation Mix**, a task-adaptive composition mechanism that learns their relative importance for each task. A pair decoder subsequently scores feasible task–machine assignments using learned compatibility together with pair-specific execution and resource information. Hard feasibility is handled by an event-driven scheduling environment, while CRST learns preferences only among feasible assignments. This separation keeps constraint enforcement exact and allows the policy to focus on the downstream quality of scheduling decisions.

Our contributions are threefold:

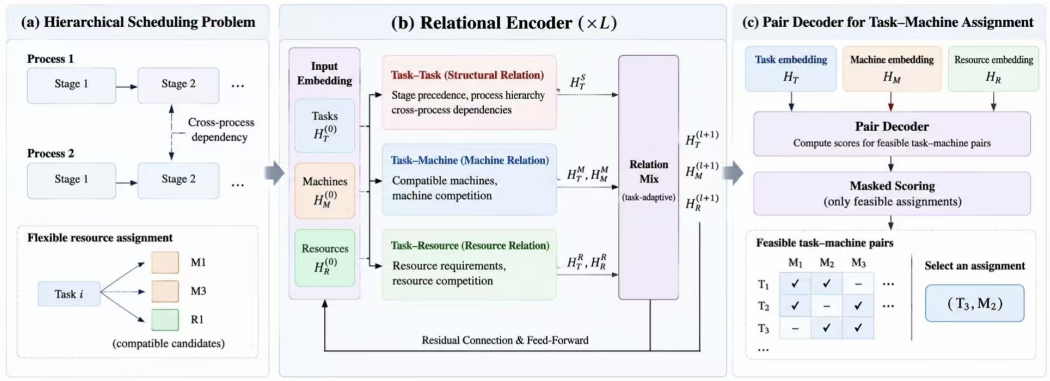
- We formulate a practically motivated hierarchical manufacturing scheduling problem with tightly coupled workflow, machine, resource, and temporal constraints, substantially extending the standard FJSP setting.
- We propose **CRST**, a heterogeneous relational scheduling framework that separately models workflow, machine, and resource interactions and adaptively composes them through *Relation Mix*, followed by pair-level task–machine decoding. An event-driven environment separates exact feasibility from learned preference. For large-scale inference, we further introduce a bounded-context decomposition that preserves global scheduling consistency while reducing dense task interactions.
- Extensive experiments demonstrate strong solution quality, generalization, and scalability. On 200-task instances, CRST achieves only a 1.12% average gap to a 30-minute CP-SAT reference; on 500- and 1000-task instances, it outperforms the same reference by 8.59% and 7.00%, respectively, while requiring only 27.48 and 35.44 seconds. On 2000-task instances, bounded-context inference reduces peak allocated GPU memory to only 5.61% of the original level, with only a 0.48% increase in makespan.

2 Related Work

Learning-based scheduling. Deep reinforcement learning has emerged as an alternative to exact optimization, metaheuristics, and manually designed dispatching rules by learning constructive scheduling policies directly from problem states. Decima (Mao et al., 2019) demonstrates the potential of graph-based RL for dependency-aware resource scheduling in data-processing clusters. For JSSP and FJSP, recent methods increasingly combine graph representations with DRL: L2D (Zhang et al., 2020) uses a disjunctive graph with PPO, (Song et al., 2023) model operation–machine relations with a heterogeneous graph, DANIEL (Wang et al., 2024) introduces dual attention for operation–machine interactions, and ReSched (Xiao et al., 2026) adopts a compact Transformer-based formulation. Other studies explore multi-action policies and heterogeneous graph representations (Lei et al., 2022; Wan et al., 2024). Despite their strong performance and cross-size generalization, these methods are primarily designed around the conventional operation–machine structure of JSSP/FJSP.

Industrial extensions of flexible job-shop scheduling. A complementary line of work extends the classical FJSP toward more realistic industrial settings without relying on learned scheduling policies. Kasapidis et al. (Kasapidis et al., 2023) study a multi-resource FJSP with arbitrary precedence graphs and formulate it using MILP and constraint programming. Subsequent work considers richer renewable and cumulative resource requirements (Kasapidis et al., 2025) and industrial settings with partially necessary resources and staged operations (Perrachon et al., 2025). Other extensions incorporate machine availability and maintenance (Gao et al., 2006), parallel and precedence-constrained operations in flexible assembly settings (Müller & Rieck, 2026), and transportation times or limited transportation resources (Karimi et al., 2017; Berterottière et al., 2024). These studies demonstrate that heterogeneous resources, non-linear workflows, and temporal operating constraints are important features of practical manufacturing scheduling. However, they are primarily addressed through mathematical programming, constraint programming, or handcrafted metaheuristics. Our work instead investigates how these heterogeneous interactions can be represented and exploited by a reusable reinforcement learning policy.

Figure 1: Overview of CRST. The hierarchical scheduling problem is encoded through task–task, task–machine, and task–resource relations. The relational encoder performs relation-specific reasoning followed by task-adaptive Relation Mix, and the pair decoder scores feasible task–machine assignments.



3 Problem Formulation

We consider a hierarchical manufacturing scheduling problem with multiple production processes. Let $\mathcal{T}$, $\mathcal{M}$, and $\mathcal{R}$ denote the sets of tasks, machines, and reusable resources, respectively. Each task $i \in \mathcal{T}$ is assigned to one compatible machine $m \in \mathcal{M}_i$ with processing time p_{im} and may additionally require reusable auxiliary resources. A schedule specifies the machine assignment, resource allocation, start time s_i, and completion time c_i of every task.

A feasible schedule must satisfy the following constraints:

- **Three-level production hierarchy.** Unlike the standard FJSP, which usually represents each job as a flat sequence of operations, our problem explicitly models production at three levels: process, stage, and task. A problem may contains multiple processes. Stages are ordered within a process, whereas tasks belonging to the same stage are mutually independent and may execute in parallel.
- **Precedence dependencies.** Stages within a process are executed in order. A task in a stage can start only after all tasks in its predecessor stage have completed. Additional precedence relations may also connect tasks belonging to different processes.
- **Machine compatibility and exclusivity.** Each task is assigned to exactly one compatible machine. A machine can process at most one task at a time. Tasks restricted to a prescribed machine are represented by $|\mathcal{M}_i| = 1$.
- **Machine rest periods.** Each machine may contain predefined rest intervals during which no processing is performed. If a task starts before a rest interval and is not completed when the rest begins, its processing is paused and resumes on the same machine after the rest period. Hence, a task may span multiple working intervals, and its calendar duration $c_i - s_i$ can exceed its nominal processing time p_{im}.
- **Reusable-resource capacity.** Some tasks require auxiliary reusable resources during processing. Resource units are exclusive while occupied and are released when the task completes. Multiple interchangeable units of the same resource type form a resource pool whose concurrent usage cannot exceed its capacity.

Resource allocation is handled by the scheduling environment rather than exposed as an additional policy decision. The learned policy selects a feasible task–machine pair, while the simulator determines a compatible physical resource allocation and the corresponding execution interval under machine rest periods. We focus on feasible schedule construction and makespan minimization. Detailed constraint formulations and exact optimization models are provided in Appendix B. These

constraints induce three complementary interactions: workflow relations among tasks, processing relations between tasks and machines, and capacity relations between tasks and reusable resources. CRST explicitly preserves these relation types in its heterogeneous scheduling representation.

4 METHOD

4.1 CONSTRUCTIVE DECISION PROCESS

We formulate schedule construction as an event-driven Markov decision process. At decision epoch t, let S_t denote the current partial schedule and T_t the corresponding physical simulation time. The index t counts scheduling decisions rather than uniformly spaced time steps. As summarized in Algorithm 1, CRST repeatedly selects a feasible task–machine pair and updates the global scheduling state.

State. The state S_t summarizes the schedule constructed so far, including task assignment and completion status, machine availability, reusable-resource availability, and the precedence-induced readiness of unfinished tasks. Static instance information, including workflow dependencies, machine compatibility, processing times, resource requirements, and machine calendars, remains fixed throughout the episode.

Feasible actions. The policy selects a task–machine assignment $a_t = (i, m)$ from the feasible action set $\mathcal{A}(S_t)$. Rather than asking the policy to learn hard constraints, the environment constructs this set exactly from the current scheduling state. In particular, a pair (i, m) is admissible only if task i is ready, machine m is compatible with i, and the assignment can start at the current simulation time under the corresponding machine, reusable-resource, and calendar constraints.

This construction separates *feasibility* from *preference*. The simulator determines whether an assignment is valid, while the policy learns to distinguish among feasible alternatives according to their downstream effects on workflow progress, machine contention, and shared resource competition. CRST therefore models a distribution only over $\mathcal{A}(S_t)$.

State transition. Given an action (i, m), the simulator determines a feasible physical-resource allocation and computes its execution interval under the joint machine and resource calendars. The assignment is then committed, after which machine and resource availability, task completion times, and the readiness of successor tasks are updated.

Decisions are event driven. If $\mathcal{A}(S_t) = \varnothing$ while unfinished tasks remain, no policy action is introduced solely to advance time. Instead, the simulator moves directly to the next event at which the scheduling state may change, such as a task becoming ready or a machine or resource becoming available, and then reconstructs the feasible action set. An episode terminates when all tasks have been scheduled, and its resulting schedule is evaluated by the makespan objective. At each decision epoch, the simulator state S_t is converted into a heterogeneous relational observation over tasks, machines, and reusable resources. The resulting representation and its CRST encoder are introduced next.

4.2 RELATIONAL ENCODER

As illustrated in Figure 1, CRST encodes the scheduling state through L relational blocks that separately reason over workflow, task–machine, and task–resource interactions. The resulting task contexts are combined by task-adaptive *Relation Mix*, while machine and ResourcePool representations are updated through the corresponding reverse interactions. This preserves heterogeneous relation semantics before composition and produces the representations used by the pair decoder for feasible task–machine assignment.

State representation. At each decision step, we represent the partial schedule as a heterogeneous graph

$$\mathcal{G}_t = (\mathcal{V}_T, \mathcal{V}_M, \mathcal{V}_R, \mathcal{E}_{TT}, \mathcal{E}_{TM}, \mathcal{E}_{TR}),$$

containing task, machine, and ResourcePool nodes. Each task, machine, and ResourcePool node is associated with a 4-, 3-, and 3-dimensional feature vector, respectively. Task–Task relations are

characterized by three discrete structural attributes, while Task–Machine and Task–Resource edges carry 2-dimensional relation-specific features. In addition, each candidate task–machine pair is associated with a 2-dimensional resource-risk descriptor that summarizes the resource-side consequence of committing the corresponding assignment.

Input embedding. Let X_v denote the input feature matrix for entity type $v \in \{T, M, R\}$, corresponding to tasks, machines, and ResourcePools. CRST maps the heterogeneous node features into a shared d-dimensional latent space using type-specific projections,

$$H_v^{(0)} = \mathrm{LN}(W_v X_v + b_v), \qquad v \in \{T, M, R\}.$$

The resulting embeddings are then updated by L relational encoder blocks. Task–Task structural attributes are incorporated as attention biases, whereas Task–Machine and Task–Resource edge features condition their corresponding bipartite attention modules.

Workflow reasoning. The workflow branch operates over the task dependency structure. For attention head a, the structural compatibility between unfinished tasks i and j is

$$e_{ij,a}^S = \frac{(q_{i,a}^S)^\top k_{j,a}^S}{\sqrt{d_a}} + b_{r_{ij},a}^{\mathrm{rel}} + b_{h_{ij},a}^{\mathrm{hop}} + b_{u_{ij},a}^{\mathrm{scope}}, \tag{1}$$

where r_{ij} denotes the dependency relation, h_{ij} the truncated graph distance, and u_{ij} whether the interaction lies within or across processes. These structural biases allow the same learned task representations to distinguish different roles in the production graph.

The branch combines a local view restricted to self, direct predecessors, and direct successors with a global view over unfinished tasks. The former preserves immediate precedence propagation, while the latter provides short communication paths between distant operations. Their exact attention and fusion equations are deferred to Appendix E.

Reciprocal capacity reasoning. Machine and resource interactions are modeled as two bidirectional bipartite relations:

$$\begin{aligned}(C_T^{M,(l)}, C_M^{T,(l)}) &= \mathrm{BiAttn}_{TM}^{(l)}(H_T^{(l)}, H_M^{(l)}, E_{TM}),\\ (C_T^{R,(l)}, C_R^{T,(l)}) &= \mathrm{BiAttn}_{TR}^{(l)}(H_T^{(l)}, H_R^{(l)}, E_{TR}).\end{aligned} \tag{2}$$

Tasks aggregate processing alternatives and reusable-resource conditions, while machines and ResourcePools aggregate prospective demand from unfinished tasks. Both relations use edge-conditioned attention, with Task–Machine edges encoding pair-specific execution information and Task–Resource edges encoding resource demand and holding-time information. Full attention details are given in Appendix E.

Relation Mix. CRST preserves workflow, machine, and resource contexts separately before task-adaptive composition. Let $C_i^{(l)} = [c_i^{S,(l)} \| c_i^{M,(l)} \| c_i^{R,(l)}]$. The mixing weights and task update are

$$\boldsymbol{\alpha}_i^{(l)} = \mathrm{softmax}\Big(f_{\mathrm{mix}}^{(l)}(C_i^{(l)})\Big), \qquad \tilde{h}_i^{T,(l+1)} = \mathrm{LN}\left(h_i^{T,(l)} + \sum_{r\in\{S,M,R\}} \alpha_{i,r}^{(l)} c_i^{r,(l)}\right). \tag{3}$$

The weights adapt the contribution of workflow, machine, and resource information for each task and layer. The softmax weights adapt the relative contribution of the three relational views for each task and layer. Machine and ResourcePool embeddings are updated from the corresponding reverse-direction contexts, followed by type-specific feed-forward modules.

4.3 Pair Decoder

Relation Mix produces task representations that integrate workflow, machine, and resource interactions, whereas a scheduling action is defined over a task–machine pair. We summarize the final entity representations as

$$g_t = \mathrm{Pool}_T(H_T^{(L)}) \| \, \mathrm{Pool}_M(H_M^{(L)}) \| \, \mathrm{Pool}_R(H_R^{(L)}), \tag{4}$$

where only unfinished tasks participate in the task pool. For each candidate pair (i, m), CRST computes

$$\ell_{im} = \frac{(W_T h_i^T)^\top (W_M h_m^M)}{\sqrt{d}} + f_{\text{pair}}(p_{im}), \tag{5}$$

with

$$p_{im} = \left[h_i^T \| h_m^M \| (h_i^T \odot h_m^M) \| \phi_{TM}(e_{im}^{TM}) \| \phi_R(\rho_{im}^R) \| g_t\right]. \tag{6}$$

Here, $\rho_{im}^R = [B_{im}^R, K_{im}^R]$ is an action-conditioned resource descriptor that summarizes potential resource blocking and the downstream criticality of affected competitors. Detailed feature definitions and decoder parameterization are provided in Appendix E.

The first term in Eq. (5) provides task–machine compatibility, while the pair MLP incorporates assignment-specific execution and resource effects. Finally, logits are normalized only over the simulator-provided feasible action set:

$$\pi_\theta(i, m \mid S_t) = \frac{\exp(\ell_{im})}{\sum_{(j,n)\in\mathcal{A}(S_t)} \exp(\ell_{jn})}, \qquad (i, m) \in \mathcal{A}(S_t). \tag{7}$$

4.4 Reward and PPO Training

CRST is trained with PPO to minimize makespan. Since policy decisions are event-driven, consecutive decisions may be separated by unequal amounts of physical time. Let T_t denote the simulation time at decision state S_t. Policy decisions terminate once all tasks have been assigned; after the final assignment, the simulator advances deterministically, without further actions, until all running tasks complete.

We therefore define the elapsed-time increment as

$$\Delta\tau_t = \begin{cases} T_{t+1} - T_t, & t < K, \\ C_{\max} - T_K, & t = K, \end{cases} \tag{8}$$

where K is the final policy decision and $C_{\max}$ is the resulting makespan. The reward is

$$r_t = -\frac{\Delta\tau_t}{LB_0} + \beta\left[\Phi(S_{t+1}) - \Phi(S_t)\right], \qquad \Phi(S) = -\frac{LB(S)}{LB_0}, \tag{9}$$

with $LB_0 = \max\{1, LB(S_0)\}$ and terminal potential $\Phi(S_{K+1}) = 0$.

With $T_0 = 0$, the elapsed-time component satisfies

$$\sum_{t=0}^{K} -\frac{\Delta\tau_t}{LB_0} = -\frac{C_{\max}}{LB_0}, \tag{10}$$

so the undiscounted return is directly aligned with makespan minimization. The potential term provides denser progress feedback and, with $\gamma = 1$, telescopes to an instance-dependent constant.

This division of responsibility ensures that every rollout uses the scheduler's explicit feasibility checks. It also restricts policy optimization to the ranking of legal assignments, reducing exploration over invalid actions. Temporarily idle states are handled by event advancement, without a learned waiting action or premature rollout termination.

5 Computational Experiments

Instance generation. We generate scheduling instances by first sampling the hierarchical production structure and then instantiating machine, resource, and temporal constraints. Each instance contains multiple processes, where each process consists of an ordered sequence of stages and each stage contains one or more parallel tasks. For every task, we sample its compatible-machine set and machine-dependent processing times, and optionally assign reusable-resource requirements. Machine rest periods are generated independently and are taken into account when computing calendar-aware task completion times. To evaluate different scheduling regimes, we vary the instance size and the strength of machine and resource contention, including the number of processes, tasks, machines, reusable resources, machine flexibility, resource capacity, and machine-rest density.

Algorithm 1: Event-Driven Schedule Construction with CRST

Input: Scheduling instance $\mathcal{I}$; trained policy π_θ
Output: Feasible schedule $\mathcal{S}$

Initialize task, machine, and resource states; $T \leftarrow 0$;
while *unfinished tasks remain* **do**
  $\mathcal{A}(S_T) \leftarrow \emptyset$;
  foreach *unscheduled task* i *and compatible machine* m **do**
    if (i, m) *satisfies precedence, machine, and resource requirements* **then**
      Compute earliest feasible execution (s_{im}, c_{im});
      if $s_{im} \leq T$ **then**
        Add (i, m) to $\mathcal{A}(S_T)$ and cache execution attributes;
  if $\mathcal{A}(S_T) \neq \emptyset$ **then**
    Build heterogeneous observation $\mathcal{G}_T$;
    Encode $\mathcal{G}_T$ with the CRST relational encoder;
    Score feasible pairs with the Pair Decoder;
    Select $(i, m) \sim \pi_\theta(\cdot \mid S_T)$ (or $\arg\max$ at evaluation);
    Commit (i, m) and update task, machine, resource, and precedence states;
  else
    Advance T to the next event that may change feasibility;
return $\mathcal{S}$;

Table 1: Mean makespan, relative gap, and runtime on four benchmark scenarios. Each scenario contains 30 instances. Gap is computed per instance relative to CP-SAT (1800 s) and then averaged.

	Resource scale		Machine multi-mode		Cross-process		Hub flexibility	
Method	Makespan (Gap %)	Time (s)	Makespan (Gap %)	Time (s)	Makespan (Gap %)	Time (s)	Makespan (Gap %)	Time (s)
CP-SAT (1800 s)	3781.03 (0.00)	1800	2141.03 (0.00)	1800	2173.93 (0.00)	1800	1978.03 (0.00)	1800
CP-SAT (60 s)	3851.20 (1.82)	60	2408.87 (12.51)	60	2361.37 (8.62)	60	2181.43 (10.35)	60
CP-SAT (120 s)	3816.80 (0.93)	120	2325.03 (8.59)	120	2307.37 (6.13)	120	2105.33 (6.52)	120
EFT	4513.80 (19.76)	0.31	2685.47 (25.48)	0.54	2618.47 (20.42)	0.50	2412.17 (22.24)	0.44
SPT	4512.23 (19.69)	0.31	2691.67 (25.67)	0.55	2656.40 (22.20)	0.51	2432.33 (23.25)	0.44
FIFO	4232.63 (12.12)	0.34	2524.50 (18.00)	0.58	2599.87 (19.56)	0.55	2378.17 (20.26)	0.49
MWKR	4086.60 (8.11)	0.52	2572.40 (20.20)	0.78	2585.13 (18.99)	0.71	2372.77 (20.09)	0.67
MOPNR	4094.60 (8.45)	0.51	2523.20 (17.84)	0.74	2584.50 (18.89)	0.69	2330.90 (17.95)	0.65
ReSched(greedy)	4214.07 (12.02)	1.46	2704.93 (26.37)	1.74	2699.33 (24.15)	1.60	2563.23 (29.72)	1.74
DANIEL(greedy)	4212.13 (11.63)	4.02	2721.57 (27.14)	3.83	2903.48 (33.54)	3.79	2664.03 (34.59)	3.67
ReSched (sample-64)	3910.57 (3.43)	49.61	2414.70 (12.78)	52.01	2474.43 (13.82)	48.03	2296.00 (16.08)	56.91
DANIEL (sample-64)	3935.99 (4.10)	55.82	2436.56 (13.80)	48.75	2463.13 (13.30)	55.60	2293.24 (15.94)	50.58
CRST (greedy)	3858.73 (2.10)	13.92	2262.97 (5.73)	13.61	2266.43 (4.30)	13.55	2081.50 (5.38)	11.94
CRST (sample-64)	**3787.23 (0.18)**	22.49	**2159.73 (0.91)**	23.30	**2188.53 (0.70)**	21.61	**2030.73 (2.69)**	18.32

Training and test instances. CRST is trained exclusively on randomly generated 200-task instances, without using instances specifically tailored to any of the evaluation scenarios. For testing, we construct four scenario-specific benchmark sets whose generative distributions are extracted from characteristic structures and constraint patterns of the target industrial scheduling problem. The resulting instances are industrial-inspired synthetic distributions rather than arbitrarily designed stress cases. Unless otherwise stated, the same trained policy is evaluated on all four test distributions without additional training or fine-tuning.

Main results. We first evaluate CRST on four 200-task benchmark scenarios with different constraint structures: Resource scale, Machine multi-mode, Cross-process, and Hub flexibility. We compare against short-budget CP-SAT, five dispatching heuristics, and two learning-based scheduling baselines. For each scenario, we report the mean makespan, the per-instance gap relative to CP-SAT (1800 s), and the average inference time. CRST is evaluated using both greedy decoding and sample-64 decoding.

Table 2: Greedy generalization results on large-scale instances. Gap is computed relative to the mean makespan of CP-SAT (1800 s).

	500 tasks		1000 tasks	
Method	Makespan (Gap %)	Time (s)	Makespan (Gap %)	Time (s)
CP-SAT (1800 s)	3638.66 (0.00)	1800.00	8270.07 (0.00)	1800.00
ReSched(greedy)	3952.83 (+8.63)	4.81	7877.93 (-4.74)	12.02
DANIEL (greedy)	4366.89 (+20.01)	12.42	8287.29 (+0.21)	19.25
CRST (greedy)	**3326.17 (-8.59)**	27.48	**7690.93 (-7.00)**	35.44

Table 3: Effect of the proposed decomposition strategy on 2000-task instances. Memory denotes PyTorch peak allocated GPU memory. Relative changes are computed per instance and averaged.

Method	Makespan	Time (s)	Peak Memory (MB)
CRST (Full Graph)	**3161.62**	53.41	691.50
CRST (Decomposed)	3174.41 $(+0.48\%)$	**40.74** (-23.72%)	**38.77** (-94.39%)

Despite being trained only on generic randomly generated 200-task instances, CRST already approaches the much more expensive CP-SAT (1800 s) reference on the four scenario-specific test sets. With sample-64 decoding, the average gap to CP-SAT (1800 s) is only 1.12%, with scenario-wise gaps of 0.18%, 0.91%, 0.70%, and 2.69%. This suggests that, at the training scale, CRST recovers schedules close to those obtained by substantially longer combinatorial search while requiring only tens of seconds.

Large-scale generalization. The large-scale results reveal a notable change in the relative performance of the two approaches. Whereas CRST remains slightly above the CP-SAT (1800 s) reference on 200-task instances, the gap becomes negative on both larger scales. Greedy CRST achieves a -8.59% gap on 500-task instances and a -7.00% gap on 1000-task instances, while requiring only 27.48 s and 35.44 s on average, respectively, compared with the fixed 1800-second CP-SAT budget.

Importantly, these results are obtained with the same policy trained only on 200-task instances. The negative gaps do not imply that CRST outperforms the optimal solution; rather, they indicate that as the problem size increases, the fixed-budget CP-SAT search becomes increasingly difficult to exploit, whereas the learned constructive policy retains strong solution quality.

Memory-efficient large scale decomposition. Direct full-graph inference becomes increasingly memory intensive as the number of tasks grows, which is particularly undesirable when multiple rollouts are required for sampling-based decoding. We therefore employ a rolling inference strategy that bounds the number of tasks processed by CRST at each planning step while maintaining a single global scheduling environment. Thus, only the neural observation is decomposed; machine, resource, precedence, and calendar states remain globally consistent.

As shown in Table 3, on 2000-task instances the rolling scheme reduces PyTorch peak allocated memory from 691.50 MB to 38.77 MB, a 94.39% reduction, while changing the average makespan by only 0.48%. This substantially increases the memory headroom available for multi-rollout sampling on large instances.

Ablation experiments The ablation study in Table 4 confirms that all three relational views contribute to CRST. Removing workflow, machine, or resource reasoning increases the overall gap from 4.38% to 12.04%, 9.39%, and 13.06%, respectively, with the largest degradation caused by removing resource reasoning. Relation Mix is also important: replacing the task-adaptive composition with uniform weights or an unconstrained MLP increases the gap to 6.26% and 7.42%. Pair-level information provides an additional benefit, while two and three relational blocks perform similarly, with three blocks giving the lowest overall gap among the tested settings. Ablations in Appendix G confirm that all three relational views, task-adaptive Relation Mix, and pair-level decoding contribute substantially to CRST.

Table 4: Ablation results on the four 200-task benchmark scenarios. Each scenario contains 30 instances. Gap is computed per instance relative to CP-SAT (1800 s) and then averaged.

Method	Resource scale Makespan (Gap %)	Machine multi-mode Makespan (Gap %)	Cross-process Makespan (Gap %)	Hub flexibility Makespan (Gap %)	Overall Makespan (Gap %)
CRST (3 blocks)	3858.73 (2.10)	**2262.97 (5.73)**	2266.43 (4.30)	2081.50 (5.38)	**2617.41 (4.38)**
1 block	3883.77 (2.74)	2388.43 (11.57)	2363.53 (8.76)	2161.80 (9.55)	2699.38 (8.16)
2 blocks	**3849.77 (1.87)**	2296.63 (7.26)	**2263.33 (4.15)**	**2081.77 (5.30)**	2622.88 (4.65)
4 blocks	3876.47 (2.60)	2260.30 (5.58)	2316.47 (6.60)	2102.77 (6.36)	2639.00 (5.29)
w/o Pair Risk	3870.93 (2.50)	2298.30 (7.37)	2377.47 (9.39)	2127.33 (7.62)	2668.51 (6.72)
Task+Machine decoder	3855.40 (2.00)	2430.47 (13.53)	2359.90 (8.59)	2247.60 (13.93)	2723.34 (9.51)
MLP decoder	3932.80 (4.11)	2413.63 (12.79)	2426.93 (11.65)	2203.50 (11.61)	2744.22 (10.04)
w/o Workflow	3984.17 (5.42)	2464.00 (15.12)	2511.30 (15.57)	2214.27 (12.05)	2793.43 (12.04)
w/o Machine	3935.27 (4.10)	2382.77 (11.35)	2409.20 (10.87)	2198.50 (11.24)	2731.43 (9.39)
w/o Resource	3975.80 (5.28)	2479.70 (15.88)	2494.13 (14.80)	2296.63 (16.27)	2811.57 (13.06)
Uniform Relation Mix	3900.43 (3.30)	2275.60 (6.36)	2335.27 (7.48)	2130.97 (7.91)	2660.57 (6.26)
MLP Relation Mix	3868.07 (2.35)	2332.23 (8.99)	2390.43 (9.99)	2139.90 (8.36)	2682.66 (7.42)

Table 5: Comparison with generic heterogeneous graph encoders on the 200-task benchmark. Gap is computed from scenario-level mean makespan relative to CP-SAT (1800 s). Lower is better.

Method	Resource scale Makespan (Gap %)	Machine multi-mode Makespan (Gap %)	Cross-process Makespan (Gap %)	Hub flexibility Makespan (Gap %)
CP-SAT (1800 s)	3781.03 (0.00)	2141.03 (0.00)	2173.93 (0.00)	1978.03 (0.00)
R-GCN	3907.47 (3.34)	2316.07 (8.18)	2353.10 (8.24)	2153.77 (8.88)
HAN	3882.10 (2.67)	2336.00 (9.11)	2334.60 (7.39)	2156.60 (9.03)
HGT	4038.00 (6.80)	2422.70 (13.16)	2424.80 (11.54)	2232.30 (12.85)
CRST	**3858.73 (2.10)**	**2262.97 (5.73)**	**2266.43 (4.30)**	**2081.50 (5.38)**

Encoder comparison. We replace the CRST relational encoder with representative heterogeneous graph architectures, including R-GCN (Schlichtkrull et al., 2018), HAN-style (Wang et al., 2019), and HGT-style (Hu et al., 2020), while keeping the observation features, action space, feasibility mask, pair decoder, critic, PPO training, and training data unchanged. All variants are trained from scratch under the same budget, allowing Table 5 to isolate the effect of the graph encoder. HAN-style and HGT-style denote scheduling-oriented adaptations of the corresponding original architectures rather than exact reproductions.

6 Conclusion

We presented CRST for a practically motivated hierarchical manufacturing scheduling problem with heterogeneous workflow, machine, resource, and temporal interactions. CRST explicitly preserves these relation types during encoding, performs task-adaptive composition through *Relation Mix*, and scores feasible task–machine assignments with a pair-level decoder, while an event-driven environment handles exact feasibility.

Across four 200-task benchmark scenarios, CRST achieves solution quality close to a 30-minute CP-SAT reference and consistently outperforms heuristic and learning-based baselines. The advantage becomes more pronounced on larger instances, where CRST maintains strong solution quality under much smaller wall-clock budgets. Encoder-replacement and ablation experiments further support the importance of scheduling-specific relational reasoning, and the proposed decomposition strategy substantially reduces memory consumption for large-scale inference. Overall, these results suggest that explicitly modeling heterogeneous scheduling relations is a promising direction for scalable learning-based optimization in complex industrial scheduling.

REFERENCES

Lucas Berterottière, Stéphane Dauzère-Pérès, and Claude Yugma. Flexible job-shop scheduling with transportation resources. *European Journal of Operational Research*, 312(3):890–909, 2024. doi: 10.1016/j.ejor.2023.07.036.

Jie Gao, Mitsuo Gen, and Linyan Sun. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing*, 17:493–507, 2006. doi: 10.1007/s10845-005-0021-x.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pp. 2704–2710, 2020. doi: 10.1145/3366423.3380027.

Sajad Karimi, Zaniar Ardalan, Bashir Naderi, and Mohammad Mohammadi. Scheduling flexible job-shops with transportation times: Mathematical models and a hybrid imperialist competitive algorithm. *Applied Mathematical Modelling*, 41:667–682, 2017. doi: 10.1016/j.apm.2016.09.022.

Gregory A. Kasapidis, Stéphane Dauzère-Pérès, Dimitris C. Paraskevopoulos, Panagiotis P. Repoussis, and Christos D. Tarantilis. On the multiresource flexible job-shop scheduling problem with arbitrary precedence graphs. *Production and Operations Management*, 32(7):2322–2330, 2023. doi: 10.1111/poms.13977.

Gregory A. Kasapidis, Dimitris C. Paraskevopoulos, Ioannis Mourtos, and Panagiotis P. Repoussis. A unified solution framework for flexible job shop scheduling problems with multiple resource constraints. *European Journal of Operational Research*, 320(3):479–495, 2025. doi: 10.1016/j.ejor.2024.08.010.

Kun Lei, Peng Guo, Wenchao Zhao, Yi Wang, Linmao Qian, Xiangyin Meng, and Liansheng Tang. A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. *Expert Systems with Applications*, 205:117796, 2022. doi: 10.1016/j.eswa.2022.117796.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the 2019 ACM SIGCOMM Conference*, pp. 270–288. ACM, 2019. doi: 10.1145/3341302.3342080.

Lars Müller and Julia Rieck. Flexible job shop scheduling with sequencing flexibility, parallel operations, and transport times. *International Journal of Production Research*, 2026. doi: 10.1080/00207543.2026.2647007.

Quentin Perrachon, Alexandru-Liviu Olteanu, Marc Sevaux, Sylvain Fréchengues, and Jean-François Kerviche. Industrial multi-resource flexible job shop scheduling with partially necessary resources. *European Journal of Operational Research*, 320(2):309–327, 2025. doi: 10.1016/j.ejor.2024.07.023.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web*, pp. 593–607. Springer, 2018.

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.

Lanjun Wan, Long Fu, Changyun Li, and Keqin Li. Flexible job shop scheduling via deep reinforcement learning with meta-path-based heterogeneous graph neural network. *Knowledge-Based Systems*, 296:111940, 2024. doi: 10.1016/j.knosys.2024.111940.

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.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pp. 2022–2032, 2019. doi: 10.1145/3308558.3313562.

Xiangjie Xiao, Cong Zhang, Wen Song, and Zhiguang Cao. RESCHED: Rethinking flexible job shop scheduling from a transformer-based architecture with simplified states. In *The Fourteenth International Conference on Learning Representations*, 2026.

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. Curran Associates, Inc., 2020.

A LLM USAGE

Large language models were used during this work to assist with manuscript organization, language and grammar refinement, LaTeX preparation, and programming. LLM assistance was also used extensively in the development of the experimental code.

The authors reviewed and revised all LLM-assisted manuscript content and code. The final experimental pipeline, configurations, reported results, and scientific conclusions were verified by the authors. The authors take full responsibility for the accuracy and integrity of the submitted work.

B DETAILED PROBLEM SPECIFICATION

This section provides the complete specification of the scheduling problem introduced in Section 3. We distinguish the production hierarchy, precedence structure, machine assignment, machine rest periods, and reusable-resource requirements. These constraints are enforced exactly by the scheduling simulator and define the feasible action set used by CRST.

B.1 HIERARCHICAL PRODUCTION STRUCTURE

A scheduling instance contains a set of production processes $\mathcal{P}$. Each process $p \in \mathcal{P}$ consists of an ordered sequence of stages

$$\mathcal{S}_p = (s_{p,1}, \ldots, s_{p,K_p}),$$

and each stage s contains a set of tasks $\mathcal{T}_{p,s}$. The complete task set is

$$\mathcal{T} = \bigcup_{p \in \mathcal{P}} \bigcup_{s \in \mathcal{S}_p} \mathcal{T}_{p,s}.$$

Tasks are the smallest dispatchable units. Tasks belonging to the same stage have no precedence relation among themselves and may therefore execute in parallel, subject to machine and reusable-resource availability. In contrast, stages belonging to the same process follow their prescribed order.

This hierarchy differs from the standard FJSP representation, in which a job is typically modeled as a single sequence of operations. Here, one process may branch into several parallel tasks within a stage before synchronizing again at the following stage.

B.2 PRECEDENCE RELATIONS

Let $\mathrm{Pred}(i)$ denote the predecessor set of task i. For consecutive stages of the same process, every task in the downstream stage must wait until all tasks in the preceding stage have completed. Therefore, for $i \in \mathcal{T}_{p,s}$ and $j \in \mathcal{T}_{p,s+1}$,

$$s_j \geq \max_{i \in \mathcal{T}_{p,s}} c_i. \tag{11}$$

The problem may additionally contain precedence relations between tasks from different processes. Let $\mathcal{E}^X \subseteq \mathcal{T} \times \mathcal{T}$ denote these cross-process dependencies. For every $(i, j) \in \mathcal{E}^X$,

$$s_j \geq c_i. \tag{12}$$

Together, the within-process and cross-process relations form the task dependency graph used by the structural branch of CRST.

B.3 MACHINE ASSIGNMENT AND PROCESSING

Let $\mathcal{M}$ denote the set of machines. Each task $i \in \mathcal{T}$ can be processed only by a compatible subset $\mathcal{M}_i \subseteq \mathcal{M}$. Its processing time may depend on the selected machine and is denoted by $p_{im} > 0$ for $m \in \mathcal{M}_i$.

Each task must be assigned to exactly one compatible machine. Using the binary assignment variable y_{im},

$$\sum_{m \in \mathcal{M}_i} y_{im} = 1, \qquad y_{im} \in \{0, 1\}. \tag{13}$$

A task with $|\mathcal{M}_i| = 1$ is a fixed-machine task. Otherwise, the scheduler must select among several alternative processing routes.

Each machine has unit processing capacity. Hence, two tasks assigned to the same machine cannot occupy it simultaneously. If $I_i = [s_i, c_i)$ denotes the calendar interval associated with task i, then for any two tasks $i \neq j$ assigned to the same machine,

$$I_i \cap I_j = \varnothing. \tag{14}$$

B.4 Machine Rest Periods

Each machine m may contain a set of predefined rest intervals $\mathcal{B}_m$. Processing is not performed during these intervals.

Importantly, a task is not required to fit entirely inside one continuous working interval. If a task starts before a machine rest period and remains unfinished when the rest begins, processing is suspended and subsequently resumes on the same machine. These interruptions are imposed by the machine calendar and are not scheduling-controlled preemptions.

Define the machine availability function

$$a_m(t) = \begin{cases} 1, & \text{machine } m \text{ is working at time } t, \\ 0, & \text{machine } m \text{ is in a rest period.} \end{cases} \tag{15}$$

If task i is assigned to machine m, its start and completion times satisfy

$$\int_{s_i}^{c_i} a_m(t)\, dt = p_{im}. \tag{16}$$

Consequently,

$$c_i - s_i \geq p_{im},$$

with strict inequality whenever processing crosses one or more machine rest periods. The simulator computes the corresponding calendar-aware completion time exactly when evaluating a candidate task–machine assignment.

B.5 Reusable Resource Constraints

Some tasks additionally require reusable auxiliary resources during processing. Physical resource units may be grouped into resource pools whose units are interchangeable for compatible tasks.

Let $\mathcal{R}$ denote the set of resource pools and let C_r denote the number of available physical units in pool r. For a task i, let q_{ir} denote the number of units required from pool r, with $q_{ir} = 0$ if the task does not use that resource type.

At every physical time τ, concurrent resource usage must satisfy

$$\sum_{i:\, \tau \in I_i} q_{ir} \leq C_r, \qquad \forall r \in \mathcal{R}. \tag{17}$$

A physical resource unit is exclusive while assigned to a task and becomes available again when that task completes. The learned policy does not select the physical unit explicitly. Instead, after the policy chooses a feasible task–machine pair (i, m), the simulator selects a compatible resource realization and commits the corresponding allocation.

Thus, resource feasibility is enforced exactly without enlarging the learned action space from task–machine pairs to task–machine–resource tuples.

B.6 Feasible Schedule and Objective

A complete schedule specifies, for every task i,

- a compatible machine assignment;
- a feasible physical-resource allocation when required;
- a start time s_i;
- and a completion time c_i.

A schedule is feasible if all hierarchical and cross-process precedence relations, machine compatibility and non-overlap conditions, machine rest periods, and reusable-resource capacities are satisfied.

Although the underlying production setting may admit multiple business objectives, this work considers makespan minimization. The objective is

$$\min C_{\max}, \qquad C_{\max} = \max_{i \in \mathcal{T}} c_i. \tag{18}$$

This specification defines the simulator state transitions and the feasible task–machine action set used throughout the paper. Exact CP and MIP encodings of these constraints are given in the following appendix.

C Exact Optimization Formulations

This section presents the exact optimization formulations used as reference solvers. The main modeling difficulty arises from machine rest periods. Unlike a standard FJSP, the calendar duration of a task is not necessarily equal to its nominal processing time: a task may cross one or more machine rest intervals, during which processing is suspended and subsequently resumed on the same machine.

We first describe a constraint-programming formulation, where machine alternatives, non-overlap, and calendar interruptions can be represented directly. We then present a discrete-time formulation used for MIP and CP-SAT, where calendar-aware completion times are precomputed exactly.

C.1 Common Notation

Let $\mathcal{T}$, $\mathcal{M}$, and $\mathcal{R}$ denote the sets of tasks, machines, and reusable-resource pools, respectively. For each task $i \in \mathcal{T}$, $\mathcal{M}_i \subseteq \mathcal{M}$ denotes its set of compatible machines, and p_{im} is the required processing time when task i is assigned to machine $m \in \mathcal{M}_i$.

Let s_i and c_i denote the start and completion times of task i. The complete precedence relation is denoted by $\mathcal{E} \subseteq \mathcal{T} \times \mathcal{T}$ and contains both within-process hierarchical dependencies and cross-process dependencies. Thus, every $(i, j) \in \mathcal{E}$ requires

$$c_i \leq s_j. \tag{19}$$

For reusable-resource pool $r \in \mathcal{R}$, let C_r denote its capacity and q_{ir} the number of units required by task i, where $q_{ir} = 0$ when task i does not require resource r.

The optimization objective is makespan minimization:

$$\min C_{\max}, \qquad C_{\max} \geq c_i, \quad \forall i \in \mathcal{T}. \tag{20}$$

C.2 Constraint Programming Formulation

For each task i, we introduce a mandatory interval variable X_i. For every compatible machine $m \in \mathcal{M}_i$, an optional interval X_{im} represents the realization in which task i is processed on machine m. Exactly one compatible realization is selected through an alternative constraint,

$$X_i = \text{alternative}\big(\{X_{im} : m \in \mathcal{M}_i\}\big). \tag{21}$$

Precedence constraints. For each $(i, j) \in \mathcal{E}$,

$$\mathrm{endOf}(X_i) \leq \mathrm{startOf}(X_j). \tag{22}$$

This formulation treats hierarchical stage precedence and cross-process dependencies uniformly through the task dependency graph.

Machine capacity. All tasks assigned to the same machine must be mutually non-overlapping:

$$\mathrm{noOverlap}\big(\{X_{im} : i \in \mathcal{T},\, m \in \mathcal{M}_i\}\big), \qquad \forall m \in \mathcal{M}. \tag{23}$$

Machine rest periods. Let $a_m(t) \in \{0, 1\}$ denote the working intensity of machine m, where $a_m(t) = 0$ during a predefined machine rest interval and $a_m(t) = 1$ otherwise. If X_{im} is present, its required processing amount is p_{im} and its completion time must satisfy

$$\int_{\mathrm{startOf}(X_{im})}^{\mathrm{endOf}(X_{im})} a_m(t)\, dt = p_{im}. \tag{24}$$

Hence, a task may begin before a rest period and remain unfinished when the rest begins. Processing is then suspended during the rest interval and resumes on the same machine afterward. The task therefore remains associated with the same machine throughout its complete calendar span. These interruptions are imposed by the machine schedule and do not constitute scheduler-controlled preemption.

Reusable resources. A reusable resource remains occupied until the corresponding task completes. For every resource pool r, concurrent demand must not exceed its capacity. Using cumulative interval demand, this can be expressed as

$$\sum_{i \in \mathcal{T}} q_{ir}\, \mathrm{pulse}(X_i) \leq C_r, \qquad \forall r \in \mathcal{R}. \tag{25}$$

The resource is therefore also reserved across machine rest periods crossed by the task and is released only at its completion time.

Objective. The makespan is

$$C_{\max} = \max_{i \in \mathcal{T}} \mathrm{endOf}(X_i), \tag{26}$$

which is minimized by the CP solver.

C.3 Discrete-Time MIP and CP-SAT Formulation

A standard fixed-duration formulation would impose $c_i = s_i + p_{im}$. This is not valid in our setting because the calendar duration of a task depends on which machine rest intervals are crossed after its start time.

We therefore use a discrete-time formulation in which the effect of machine rests is incorporated before optimization.

Let

$$\mathcal{H} = \{0, \ldots, H\}$$

denote the discrete scheduling horizon. For every task i, compatible machine m, and admissible start time t, we precompute

$$F_{im}(t), \tag{27}$$

where $F_{im}(t)$ is the exact completion time obtained by starting task i on machine m at time t, accumulating exactly p_{im} units of effective processing, and skipping all machine rest periods encountered during execution.

Let $\mathcal{H}_{im} \subseteq \mathcal{H}$ denote the admissible starting times for pair (i, m). We introduce the binary variable

$$x_{imt} = \begin{cases} 1, & \text{if task } i \text{ starts on machine } m \text{ at time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{28}$$

Algorithm 2: CP Model Construction

Input: Instance $\mathcal{I}$ with tasks, machines, precedence relations, resource pools, processing times, and machine-rest calendars
Output: Minimum-makespan feasible schedule

Initialize CP model $\mathcal{C}$;
foreach *task* $i \in \mathcal{T}$ **do**
  Create master interval X_i;
  foreach *compatible machine* $m \in \mathcal{M}_i$ **do**
    Compute the calendar-aware completion mapping $F_{im}(t)$;
    Set the elapsed duration $D_{im}(t) \leftarrow F_{im}(t) - t$;
    Group start times with identical $D_{im}(t)$ into compact domains;
    Create optional interval X_{im} with the corresponding start-dependent duration;
  Enforce exactly one machine assignment for task i;
foreach *precedence arc* $(i, j) \in \mathcal{E}$ **do**
  Enforce $\mathrm{end}(X_i) \le \mathrm{start}(X_j)$;
foreach *machine* $m \in \mathcal{M}$ **do**
  Add a non-overlap constraint over $\{X_{im} : i \in \mathcal{T},\, m \in \mathcal{M}_i\}$;
foreach *resource pool* $r \in \mathcal{R}$ **do**
  Add the cumulative-capacity constraint $\sum_i q_{ir}\ \mathrm{pulse}(X_i) \le C_r$;
Set $C_{\max} \leftarrow \max_{i \in \mathcal{T}} \mathrm{end}(X_i)$;
Minimize $C_{\max}$;
Solve $\mathcal{C}$ with CP-SAT and extract task assignments and start/completion times;
return the best feasible schedule found within the time limit;

Assignment. Each task must select exactly one compatible machine and one admissible starting time:

$$\sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{H}_{im}} x_{imt} = 1, \qquad \forall i \in \mathcal{T}. \tag{29}$$

Its start and completion times are then

$$s_i = \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{H}_{im}} t\, x_{imt}, \tag{30}$$

$$c_i = \sum_{m \in \mathcal{M}_i} \sum_{t \in \mathcal{H}_{im}} F_{im}(t)\, x_{imt}. \tag{31}$$

Precedence constraints. For every precedence arc $(i, j) \in \mathcal{E}$,

$$c_i \le s_j. \tag{32}$$

Machine capacity. If $x_{imt} = 1$, task i reserves machine m over its complete calendar span

$$[t, F_{im}(t)).$$

Importantly, this span also contains any machine rest periods crossed by the task, because the task resumes on the same machine afterward.

Therefore, for every machine m and physical time τ,

$$\sum_{i \in \mathcal{T}} \sum_{\substack{t \in \mathcal{H}_{im}: \\ t \le \tau < F_{im}(t)}} x_{imt} \le 1. \tag{33}$$

This prevents another task from being inserted onto the same machine while an unfinished task is suspended by a machine rest period.

Reusable-resource capacity. Reusable resources are occupied over the same committed calendar span and are released only when the task completes. Thus, for each resource pool $r \in \mathcal{R}$ and time τ,

$$\sum_{i \in \mathcal{T}} \sum_{m \in \mathcal{M}_i} \sum_{\substack{t \in \mathcal{H}_{im}: \\ t \le \tau < F_{im}(t)}} q_{ir} x_{imt} \le C_r. \tag{34}$$

Makespan. Finally,

$$C_{\max} \ge c_i, \qquad \forall i \in \mathcal{T}, \tag{35}$$

and the optimization objective is

$$\min C_{\max}. \tag{36}$$

C.4 Machine-Rest Modeling in CP-SAT

Machine rest periods require special treatment in CP-SAT. In a standard flexible job-shop formulation, the duration of task i on machine m is fixed, such that

$$c_i = s_i + p_{im}.$$

This relation is no longer valid in our setting. A task may start before a machine rest period, suspend processing during the rest, and resume on the same machine afterward. Consequently, its wall-clock duration depends on its start time.

A straightforward approach would be to represent machine rests as fixed intervals and include them together with task intervals in a `NoOverlap` constraint. This, however, would forbid a task interval from spanning a rest period and therefore does not reproduce our pause-and-resume semantics. Instead, we incorporate the machine calendar directly into the duration of each task interval.

Calendar-aware completion time. For task i processed on machine m, let

$$F_{im}(t)$$

denote its completion time when processing starts at time t. $F_{im}(t)$ is obtained by accumulating p_{im} units of effective processing while skipping all rest periods of machine m. Equivalently, the corresponding wall-clock duration is

$$D_{im}(t) = F_{im}(t) - t. \tag{37}$$

For example, suppose a machine works over $[0, 5)$ and $[8, \infty)$, with a rest period over $[5, 8)$, and a task requires five units of processing. If the task starts at $t = 3$, it processes for two units before the rest and completes the remaining three units after the machine resumes. Hence,

$$F_{im}(3) = 11, \qquad D_{im}(3) = 8,$$

although its nominal processing time is only $p_{im} = 5$.

Compressed start–duration domains. Directly introducing one Boolean variable for every possible start time would lead to a large time-indexed model. We instead exploit the fact that $D_{im}(t)$ is piecewise constant over integer start times.

For each distinct wall-clock duration δ_{imk}, we collect all start times producing that duration into the domain

$$\mathcal{D}_{imk} = \{t : D_{im}(t) = \delta_{imk}\}. \tag{38}$$

Consecutive feasible start times are merged into intervals, so $\mathcal{D}_{imk}$ is represented as a compact union of integer ranges.

For example, the calendar mapping may be compressed as

$$D_{im}(t) = \begin{cases} 5, & t \in [0, 1] \cup [8, 10], \\ 8, & t \in [2, 4]. \end{cases}$$

Rather than storing every $(t, F_{im}(t))$ pair separately, the CP-SAT model therefore only needs to distinguish the corresponding duration regimes.

CP-SAT encoding. For each optional task–machine realization, we create an interval with start s_{im}, end e_{im}, variable size d_{im}, and presence variable z_{im}. For every calendar-duration regime k, an auxiliary Boolean variable y_{imk} selects the corresponding start domain:

$$\sum_k y_{imk} = z_{im}, \tag{39}$$

$$y_{imk} = 1 \Rightarrow s_{im} \in \mathcal{D}_{imk}, \tag{40}$$

$$y_{imk} = 1 \Rightarrow d_{im} = \delta_{imk}, \tag{41}$$

$$e_{im} = s_{im} + d_{im}. \tag{42}$$

In the implementation, these constraints correspond to a variable-size `NewOptionalIntervalVar`, together with `AddLinearExpressionInDomain` for the admissible start ranges and conditional equality constraints for the associated wall-clock duration.

Machine occupancy across rests. The resulting interval spans the complete calendar interval

$$[s_{im}, e_{im}),$$

including any machine rest periods crossed by the task. All optional intervals assigned to the same machine are then included in a standard `AddNoOverlap` constraint.

This is important for preserving the intended semantics. Although no effective processing occurs during a machine rest, the unfinished task remains bound to the same machine and resumes on that machine afterward. Therefore, another task cannot be inserted into the machine during the intervening rest period.

The calendar-domain construction preserves every feasible integer (s_{im}, e_{im}) pair generated by the machine calendar. The compression only groups start times that induce the same wall-clock duration and therefore does not approximate the underlying pause-and-resume behavior. For the integer-time instances considered in our experiments, the resulting CP-SAT representation is exact.

D Event-Driven Scheduling Environment

This section describes the event-driven simulator used to construct schedules and interact with the learned policy. The environment separates hard feasibility from learned preference: all precedence, machine, resource, and machine-rest constraints are enforced by the simulator, while the policy only ranks task–machine pairs that are feasible at the current decision epoch.

Unlike a fixed-step environment, the simulator does not advance physical time after every policy action. Multiple assignments may therefore be committed at the same physical time whenever they use mutually available machines and resources. Physical time advances only when no further assignment can be started at the current time.

D.1 Simulation State and Decision Epochs

Let T denote the current physical simulation time. We distinguish T from the policy decision index t: consecutive policy decisions need not be separated by a positive amount of physical time.

The simulator state S_t contains the current partial schedule together with the scheduling status required to determine future feasibility, including

- the scheduling and completion status of all tasks;
- the completion times required to evaluate task precedence;
- the current assignment and availability state of each machine;
- the occupancy and availability of reusable physical resource units;
- the machine-rest schedules; and
- the current simulation time T.

The state S_t is the complete simulator state and should be distinguished from the heterogeneous graph observation $\mathcal{G}_t$ consumed by CRST. The latter is deterministically constructed from S_t and contains only the features and relations required by the policy.

A decision epoch occurs whenever at least one unscheduled task–machine pair can begin execution at the current physical time T.

D.2 Task Readiness

A task is structurally ready only when all of its required predecessors have completed. Let $\mathrm{Pred}(i)$ denote the complete predecessor set of task i, including both hierarchical dependencies and cross-process dependencies. We define

$$\tau_i^{\mathrm{pred}} = \max_{j \in \mathrm{Pred}(i)} c_j, \tag{43}$$

with $\tau_i^{\mathrm{pred}} = 0$ when i has no predecessor.

Hence, task i can be considered for scheduling at time T only if

$$\tau_i^{\mathrm{pred}} \le T. \tag{44}$$

For the three-level production hierarchy, this condition naturally implements stage synchronization: tasks in a downstream stage become ready only after the required tasks in the preceding stage have completed.

D.3 Feasible Task–Machine Pairs

At each decision epoch, the environment constructs a feasible action set

$$\mathcal{A}(S_t) = \{(i,m) : i \text{ is unscheduled and ready},\ m \in \mathcal{M}_i,\ (i,m) \text{ can start at } T\}. \tag{45}$$

For every unscheduled task i and compatible machine m, the simulator evaluates the assignment in three stages.

First, the task must satisfy all structural precedence constraints. Second, machine m must be available for the assignment and the current time must be a valid working time of that machine. Third, if the task requires reusable resources, the simulator must be able to construct at least one compatible physical-resource allocation.

We denote by

$$(s_{im}^{\mathrm{EFT}}, c_{im}^{\mathrm{EFT}})$$

the earliest feasible execution of candidate pair (i,m) under the current partial schedule. The pair is exposed to the policy only when

$$s_{im}^{\mathrm{EFT}} \le T. \tag{46}$$

Since the simulator never constructs an assignment in the past, Eq. (46) is equivalent in practice to $s_{im}^{\mathrm{EFT}} = T$.

This distinction is important: a pair that will become feasible later is not presented as an action at the current decision epoch. Instead, the simulator advances to the appropriate future event when no pair can currently start.

D.4 Simulator-Side Resource Allocation

The policy action contains only a task and a machine,

$$a_t = (i,m). \tag{47}$$

Concrete reusable-resource units are deliberately excluded from the learned action space. For a candidate pair (i,m), the environment enumerates or constructs compatible resource realizations and retains the pair only if at least one realization is feasible.

After the policy selects (i, m), the simulator commits one feasible physical resource allocation associated with that pair. The selected units remain occupied until task i completes and are then returned to their corresponding resource pools.

This design keeps the learned action space at the task–machine level while preserving exact resource feasibility at the simulator level.

D.5 Calendar-Aware Task Execution

Machine-rest periods affect the calendar duration of a selected assignment. Suppose task i is assigned to machine m at physical time T. Processing begins at T and accumulates only while machine m is working.

Let $a_m(\tau) \in \{0, 1\}$ denote the working indicator of machine m. The simulator determines the completion time c_i from

$$\int_T^{c_i} a_m(\tau)\, d\tau = p_{im}. \tag{48}$$

If a machine rest begins before the required processing is completed, execution is suspended during the rest and resumes on the same machine afterward. Thus,

$$c_i - T \geq p_{im},$$

and the inequality is strict whenever the assignment crosses one or more machine-rest intervals.

The machine remains committed to the task over the complete calendar span $[T, c_i)$, and reusable resources allocated to the task are also retained until c_i. Machine rests therefore do not create intermediate scheduling decisions for the interrupted task.

D.6 State Transition After an Action

Once action $a_t = (i, m)$ has been selected, the environment performs the following transition:

1. commit task i to machine m;
2. commit its simulator-selected physical-resource realization;
3. compute its calendar-aware completion time c_i;
4. reserve the selected machine and resources until c_i;
5. record the assignment in the partial schedule; and
6. update the information required to determine future task readiness and resource availability.

Importantly, physical time does not necessarily change after this transition. The simulator reconstructs $\mathcal{A}(S_{t+1})$ at the same value of T. If another feasible task can begin immediately on another available machine with compatible resources, the policy is invoked again.

Consequently, several tasks can be dispatched sequentially by the policy while sharing the same physical start time.

D.7 Event Advancement

If

$$\mathcal{A}(S_t) = \varnothing \tag{49}$$

while unfinished tasks remain, no policy action is requested. Instead, the environment advances physical time directly to the earliest future event that can change scheduling feasibility.

Such events include, for example,

- completion of a running task, which releases its machine and reusable resources and may activate successor tasks; and
- the end of a machine-rest interval, after which an otherwise idle machine may again become usable.

Algorithm 3: Event-Driven Schedule Construction

Input: Scheduling instance $\mathcal{I}$ and trained policy π_θ
Output: Complete feasible schedule $\mathcal{S}$

Initialize task, machine, and resource states; set $T \leftarrow 0$;
while *unfinished tasks remain* **do**
  Construct the set of structurally ready unscheduled tasks;
  $\mathcal{A}(S_T) \leftarrow \emptyset$;
  foreach *ready task* i **do**
    foreach *compatible machine* $m \in \mathcal{M}_i$ **do**
      Check whether a compatible resource allocation exists;
      if *machine, resource, and calendar conditions permit* (i, m) **then**
        Compute the earliest feasible execution $(s_{im}^{\mathrm{EFT}}, c_{im}^{\mathrm{EFT}})$;
        if $s_{im}^{\mathrm{EFT}} \leq T$ **then**
          Add (i, m) to $\mathcal{A}(S_T)$ and cache its feasible execution information;
  if $\mathcal{A}(S_T) \neq \emptyset$ **then**
    Construct the heterogeneous observation $\mathcal{G}_T$;
    Encode $\mathcal{G}_T$ with CRST;
    Score all $(i, m) \in \mathcal{A}(S_T)$ with the Pair Decoder;
    Select $(i, m) \sim \pi_\theta(\cdot \mid S_T)$ or take $\arg\max$ during greedy evaluation;
    Commit (i, m) with the simulator-selected resource allocation;
    Update task, machine, resource, precedence, and calendar states;
  else
    Advance T to the next feasibility-changing event;
    Process all events occurring at the new physical time;
return $\mathcal{S}$;

Let $\mathcal{E}_{\mathrm{next}}(S_t)$ denote the set of such future event times. The simulator updates

$$T \leftarrow \min \{\tau \in \mathcal{E}_{\mathrm{next}}(S_t) : \tau > T\}. \tag{50}$$

It then processes all state changes occurring at the new physical time and reconstructs the feasible action set.

No explicit WAIT action is included in the policy. Waiting is therefore not a learned decision: it occurs only when the current state admits no immediately executable task–machine pair.

D.8 EVENT-DRIVEN SCHEDULE CONSTRUCTION

Algorithm 3 summarizes the complete environment logic.

D.9 ENVIRONMENT–POLICY SEPARATION

The resulting interaction has a clear division of responsibility. The environment determines which decisions are physically and structurally feasible, including precedence satisfaction, machine compatibility, reusable resource allocation, and machine-rest execution. CRST receives only the resulting feasible task–machine candidates and learns their relative preference.

The environment therefore enforces

$$\boxed{\text{feasibility}},$$

while the learned policy models

$$\boxed{\text{preference among feasible assignments}}.$$

This separation guarantees that every schedule generated by the policy satisfies the hard constraints defined in Appendix B, independently of the learned action scores.

E CRST ARCHITECTURE DETAILS

This section provides the implementation details of the CRST architecture introduced in Section 4. CRST consists of three main components: type-specific input projections, a stack of heterogeneous relational encoder blocks, and a task–machine pair decoder. The encoder preserves workflow, machine, and reusable-resource interactions as separate relational views before performing task-adaptive Relation Mix.

Unless otherwise stated, the implementation uses hidden dimension $d = 128$, 8 attention heads, $L = 3$ CRST encoder blocks, and a feed-forward hidden dimension of 256.

E.1 INPUT PROJECTION

Let

$$X_T \in \mathbb{R}^{N_T \times d_T}, \qquad X_M \in \mathbb{R}^{N_M \times d_M}, \qquad X_R \in \mathbb{R}^{N_R \times d_R}$$

denote the task, machine, and ResourcePool feature matrices, respectively. The current implementation uses $d_T = 4$, $d_M = 3$, and $d_R = 3$.

Each entity type is independently projected into the common d-dimensional latent space:

$$H_v^{(0)} = \mathrm{LN}\left(X_v W_v^\top + b_v\right), \qquad v \in \{T, M, R\}. \tag{51}$$

Using separate input projections avoids imposing a shared interpretation on features belonging to semantically different scheduling entities.

E.2 WORKFLOW ATTENTION

The workflow branch operates only over task nodes and explicitly conditions attention on the precedence structure. For each ordered task pair (i, j), the observation provides three structural attributes:

- a relation type r_{ij} describing whether j is the query task itself, a direct predecessor or successor, an ancestor or descendant, or unrelated;
- a truncated graph distance h_{ij}; and
- a scope indicator u_{ij} distinguishing intra-process and cross-process relations.

For attention head a, the structural attention logit is

$$e^S_{ij,a} = \frac{(q^S_{i,a})^\top k^S_{j,a}}{\sqrt{d_h}} + b^{\mathrm{rel}}_{r_{ij},a} + b^{\mathrm{hop}}_{h_{ij},a} + b^{\mathrm{scope}}_{u_{ij},a}, \tag{52}$$

where $d_h = d/H$ is the dimension of each attention head.

CRST computes both a local and a global workflow context. The local branch restricts attention to the task itself and its direct predecessors and successors:

$$\mathcal{N}^{\mathrm{loc}}_i = \{i\} \cup \mathrm{Pred}_1(i) \cup \mathrm{Succ}_1(i). \tag{53}$$

The global branch uses the same structurally biased attention but considers all unfinished task nodes.

Let c^{loc}_i and c^{glo}_i denote the two resulting contexts. They are combined using a feature-wise sigmoid gate

$$g^S_i = \sigma\left(W_g[c^{\mathrm{loc}}_i \| c^{\mathrm{glo}}_i \| h^T_i] + b_g\right), \tag{54}$$

and the workflow representation is

$$c^S_i = \mathrm{LN}\left(h^T_i + g^S_i \odot c^{\mathrm{loc}}_i + (1 - g^S_i) \odot c^{\mathrm{glo}}_i\right). \tag{55}$$

The local branch therefore emphasizes immediate precedence interactions, whereas the global branch enables information exchange between structurally distant unfinished tasks.

E.3 Edge-Conditioned Bipartite Attention

Task–Machine and Task–Resource interactions are represented by two independent bidirectional attention modules. We describe their common implementation using a generic bipartite relation between left-side task node i and right-side capacity node j, where the right side is either a machine or a ResourcePool.

Let e_{ij} denote the corresponding edge feature vector. It is first encoded as

$$z_{ij} = W_{e,2}\,\mathrm{SiLU}\left(W_{e,1}e_{ij} + b_{e,1}\right) + b_{e,2}. \tag{56}$$

The edge embedding contributes independently to the attention key, value, and score. For head a, the task-side attention logit is

$$e^{L\leftarrow R}_{ij,a} = \frac{(q^L_{i,a})^\top \left(k^R_{j,a} + k^E_{ij,a}\right)}{\sqrt{d_h}} + s^E_{ij,a}, \tag{57}$$

where $k^E_{ij,a}$ and $s^E_{ij,a}$ are learned transformations of the edge embedding.

The corresponding message value is

$$v^{L\leftarrow R}_{ij,a} = v^R_{j,a} + v^E_{ij,a}, \tag{58}$$

and the task-side context is obtained by masked multi-head attention:

$$c^{L\leftarrow R}_{i,a} = \sum_{j\in\mathcal{N}^R_i} \mathrm{softmax}_j\left(e^{L\leftarrow R}_{ij,a}\right) v^{L\leftarrow R}_{ij,a}. \tag{59}$$

Only valid Task–Machine or Task–Resource edges of unfinished tasks participate in the corresponding attention operation.

The reverse direction is computed simultaneously. Machines and ResourcePools attend to the unfinished tasks connected to them using the same edge-conditioned representation. A learned self message is additionally included in the reverse attention, allowing a capacity node to preserve its own state when aggregating prospective task demand.

Applying this construction to the two relation types yields

$$(C^M_T, C^T_M) = \mathrm{BiAttn}_{TM}(H_T, H_M, E_{TM}), \tag{60}$$

$$(C^R_T, C^T_R) = \mathrm{BiAttn}_{TR}(H_T, H_R, E_{TR}). \tag{61}$$

The two modules do not share edge parameters. In particular, the final CRST model uses two-dimensional Task–Machine and Task–Resource edge features, so processing relations and reusable-resource relations retain separate parameterizations.

E.4 Relation Mix and Block Update

At encoder layer l, each task obtains three d-dimensional relational representations,

$$c^{S,(l)}_i, \qquad c^{M,(l)}_i, \qquad c^{R,(l)}_i.$$

CRST first concatenates the three contexts and predicts three relation coefficients:

$$\boldsymbol{\alpha}^{(l)}_i = \mathrm{softmax}\left(W^{(l)}_{\mathrm{mix}}[c^{S,(l)}_i \| c^{M,(l)}_i \| c^{R,(l)}_i] + b^{(l)}_{\mathrm{mix}}\right). \tag{62}$$

Thus,

$$\boldsymbol{\alpha}^{(l)}_i = [\alpha^{(l)}_{i,S}, \alpha^{(l)}_{i,M}, \alpha^{(l)}_{i,R}], \qquad \sum_{r\in\{S,M,R\}} \alpha^{(l)}_{i,r} = 1.$$

The mixed task context is

$$c^{\mathrm{mix},(l)}_i = \alpha^{(l)}_{i,S} c^{S,(l)}_i + \alpha^{(l)}_{i,M} c^{M,(l)}_i + \alpha^{(l)}_{i,R} c^{R,(l)}_i. \tag{63}$$

The task representation is updated by

$$\bar{h}_i^{T,(l+1)} = \mathrm{LN}\left(h_i^{T,(l)} + c_i^{\mathrm{mix},(l)}\right). \tag{64}$$

Machine and ResourcePool nodes are updated using the reverse contexts:

$$\bar{h}_m^{M,(l+1)} = \mathrm{LN}\left(h_m^{M,(l)} + c_m^{T,(l)}\right), \tag{65}$$

$$\bar{h}_r^{R,(l+1)} = \mathrm{LN}\left(h_r^{R,(l)} + c_r^{T,(l)}\right). \tag{66}$$

Each entity type subsequently applies an independent residual feed-forward network

$$\mathrm{FFN}_v(h) = h + W_{v,2}\,\mathrm{GELU}\left(W_{v,1}\,\mathrm{LN}(h) + b_{v,1}\right) + b_{v,2}, \tag{67}$$

where the intermediate dimension is d_{ff}.

Therefore,

$$H_T^{(l+1)} = \mathrm{FFN}_T(\bar{H}_T^{(l+1)}), \tag{68}$$

$$H_M^{(l+1)} = \mathrm{FFN}_M(\bar{H}_M^{(l+1)}), \tag{69}$$

$$H_R^{(l+1)} = \mathrm{FFN}_R(\bar{H}_R^{(l+1)}). \tag{70}$$

Stacking L blocks allows interactions to propagate across heterogeneous multi-hop paths. For example, after multiple layers a task can indirectly receive information from other tasks through Task–Machine–Task or Task–ResourcePool–Task paths while preserving the semantic distinction between those two forms of competition.

E.5 Global State Representation

After the final encoder block, CRST obtains

$$H_T^{(L)}, \qquad H_M^{(L)}, \qquad H_R^{(L)}.$$

Each node type is summarized independently using learned attention pooling. For a set of node embeddings $\{h_j\}$, the pooling weights are

$$\omega_j = \frac{\exp(w_p^\top h_j + b_p)}{\sum_{k\in\mathcal{V}} \exp(w_p^\top h_k + b_p)}, \tag{71}$$

and

$$\mathrm{Pool}(H) = \sum_{j\in\mathcal{V}} \omega_j h_j. \tag{72}$$

Only unfinished tasks participate in the task pool. Machine and ResourcePool representations are pooled over all corresponding nodes. The global scheduling representation is

$$g_t = g_T \| g_M \| g_R \in \mathbb{R}^{3d}. \tag{73}$$

E.6 Pair Decoder

A policy action corresponds to a task–machine pair rather than to an individual node. CRST therefore combines representation-level matching with explicit candidate-level context.

For candidate pair (i, m), the compatibility component is

$$\ell_{im}^{\mathrm{comp}} = \frac{(W_T h_i^T)^\top (W_M h_m^M)}{\sqrt{d}}. \tag{74}$$

The Task–Machine edge descriptor is independently embedded as

$$z_{im}^{TM} = \phi_{TM}(e_{im}^{TM}) \in \mathbb{R}^d, \tag{75}$$

where ϕ_{TM} is a two-layer MLP with a SiLU activation.

The final model additionally associates each candidate pair with a two-dimensional action-conditioned resource-risk descriptor

$$\rho_{im}^R = [B_{im}^R, K_{im}^R].$$

The network maps it to

$$z_{im}^R = \phi_R(\rho_{im}^R) = W_{R,2}\,\mathrm{SiLU}\left(W_{R,1}\rho_{im}^R + b_{R,1}\right) + b_{R,2} \in \mathbb{R}^d. \tag{76}$$

The complete pair representation is

$$p_{im} = \left[h_i^T \| h_m^M \| (h_i^T \odot h_m^M) \| z_{im}^{TM} \| z_{im}^R \| g_t\right]. \tag{77}$$

Since the first five components each have dimension d and $g_t \in \mathbb{R}^{3d}$,

$$p_{im} \in \mathbb{R}^{8d}.$$

The contextual residual is produced by

$$\ell_{im}^{\mathrm{ctx}} = W_{p,2}\,\mathrm{GELU}\left(W_{p,1}p_{im} + b_{p,1}\right) + b_{p,2}. \tag{78}$$

The final pair logit is

$$\ell_{im} = \ell_{im}^{\mathrm{comp}} + \ell_{im}^{\mathrm{ctx}}. \tag{79}$$

An important implementation detail is that the final decoder does *not* directly concatenate the last-layer Task–Resource message. Resource-side relational information reaches the decision through the Relation Mix task representation and the pooled global ResourcePool state, while the pair decoder receives only the explicitly action-conditioned resource-risk descriptor. This avoids duplicating the same relational message at both the encoder and decoder levels.

E.7 Feasibility Mask and Policy

CRST scores only simulator-provided feasible task–machine pairs. Let $M_{im} \in \{0, 1\}$ denote the action mask. The masked logit is

$$\widetilde{\ell}_{im} = \begin{cases} \ell_{im}, & M_{im} = 1, \\ -\infty, & M_{im} = 0. \end{cases} \tag{80}$$

The policy is then

$$\pi_\theta(i, m \mid S_t) = \mathrm{softmax}_{(i,m)}\left(\widetilde{\ell}_{im}\right). \tag{81}$$

During training, an action is sampled from this categorical distribution. During deterministic evaluation, the feasible pair with the largest logit is selected.

E.8 Value and Auxiliary Heads

The critic operates on the same global representation g_t used by the pair decoder:

$$V_\theta(S_t) = W_{V,2}\,\mathrm{GELU}\left(W_{V,1}g_t + b_{V,1}\right) + b_{V,2}. \tag{82}$$

The implementation also attaches three auxiliary regression heads to g_t, corresponding to workflow, resource, and calendar-related state summaries. Each auxiliary head consists of a $3d \to d \to 1$ MLP. These heads are used only to provide additional representation-learning signals during training and do not modify the policy action space or the feasibility mask.

F Training Details

CRST is trained end-to-end with Proximal Policy Optimization (PPO). Because the scheduling environment is event-driven, consecutive policy decisions are not necessarily separated by equal amounts of physical time. A rollout therefore consists of decision transitions (S_t, a_t, r_t, S_{t+1}), where t indexes policy decisions rather than uniform simulation steps. The actor selects only from the feasible task–machine pairs provided by the environment, while the critic estimates the value of the current scheduling state.

F.1 Auxiliary Supervision

The pooled state representation g_t is shared by the critic and three lightweight auxiliary prediction heads. These heads predict normalized signals describing (i) scheduling progress, (ii) reusable-resource availability, and (iii) machine availability. Denoting the corresponding simulator-derived targets by y_t^{prog}, y_t^{res}, and y_t^{mach}, the auxiliary objective is

$$\mathcal{L}_{\mathrm{aux}} = \ell(\hat{y}_t^{\mathrm{prog}}, y_t^{\mathrm{prog}}) + \ell(\hat{y}_t^{\mathrm{res}}, y_t^{\mathrm{res}}) + \ell(\hat{y}_t^{\mathrm{mach}}, y_t^{\mathrm{mach}}), \tag{83}$$

where $\ell(\cdot, \cdot)$ denotes the prediction loss used for the corresponding normalized target.

These heads provide additional representation-learning signals only during training. They neither alter the action logits nor participate in the feasibility mask at inference time.

The complete optimization objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{actor}} + \lambda_V \mathcal{L}_V - \lambda_H \mathcal{H} + \lambda_{\mathrm{aux}} \mathcal{L}_{\mathrm{aux}}. \tag{84}$$

G Ablation Study Details

We conduct controlled ablations on the four 200-task benchmark scenarios to isolate the contribution of the main architectural components of CRST. Each scenario contains 30 instances, resulting in 120 test instances in total. Unless otherwise specified, all variants use the same training and evaluation protocol as the full model. The reported gap is computed for each instance relative to the corresponding CP-SAT solution obtained with a 1800-second time limit and is then averaged.

G.1 Encoder Depth

We first vary the number of relational encoder blocks while keeping the remaining architecture unchanged. The full CRST uses three blocks. Reducing the depth to one block increases the overall gap from 4.38% to 8.16%, while two blocks achieve 4.65%. Increasing the depth to four blocks also slightly degrades performance to 5.29%.

These results suggest that a single relational layer is insufficient to propagate the coupled workflow, machine, and resource information required for effective scheduling. Increasing the depth beyond three blocks, however, does not provide additional benefits. The three-block configuration therefore offers the best overall trade-off among the evaluated depths, although the best configuration on individual scenarios is not always identical.

G.2 Pair Decoder Ablations

We next evaluate the information used by the task–machine pair decoder.

Without pair risk. The *w/o Pair Risk* variant removes the action-conditioned resource-risk descriptor from pair scoring while retaining the encoded task, machine, relation, and global-state representations. The overall gap increases from 4.38% to 6.72%, with the largest degradation observed on the Cross-process scenario. This indicates that explicitly representing assignment-dependent downstream resource interactions provides complementary information beyond the node embeddings alone.

Task–Machine decoder only. This variant restricts the decoder to task and machine representations and removes the additional assignment-specific relational context used by the full pair decoder. Its overall gap increases to 9.51%, showing that deciding from isolated task and machine representations is insufficient for accurately evaluating the consequences of a particular assignment.

MLP decoder. The structured pair-scoring mechanism is replaced by a generic MLP-based decoder. This variant obtains an overall gap of 10.04%, further supporting the use of explicit pair-level compatibility and assignment-specific features when scoring feasible task–machine actions.

G.3 Relation Ablations

To determine whether the three relational views provide complementary information, we remove each reasoning branch individually while leaving the remaining architecture unchanged.

Removing workflow reasoning increases the overall gap from 4.38% to 12.04%. Removing machine reasoning increases the gap to 9.39%, while removing resource reasoning causes the largest degradation, reaching 13.06%. The degradation is consistent across the four test scenarios.

These results support the central design of CRST: workflow, machine, and resource interactions cannot be treated as interchangeable signals. Each relation captures a different mechanism through which a scheduling decision affects the future state, and all three are required for robust performance under heterogeneous constraints.

G.4 Relation Mix Ablations

Finally, we examine how the three relation-specific contexts should be combined.

Uniform Relation Mix. We replace the learned task-adaptive coefficients with uniform weights over workflow, machine, and resource contexts. The overall gap increases from 4.38% to 6.26%. Although all three relational views remain available, their fixed contribution prevents the model from adapting to tasks dominated by different constraint mechanisms.

MLP Relation Mix. We replace the normalized task-adaptive Relation Mix with a generic MLP fusion of the three relation-specific contexts. This produces an overall gap of 7.42%. The result suggests that explicitly maintaining relation-specific contexts and allowing them to compete through normalized mixing is more effective than unconstrained feature fusion.

Taken together, the ablation results show that the performance of CRST does not arise from network depth alone. Its strongest configuration requires (i) separate workflow, machine, and resource reasoning, (ii) task-adaptive Relation Mix, and (iii) pair-level decoding that incorporates assignment-specific information.

H Bounded-Context Decomposition for Large-Scale Inference

Although CRST can in principle process a variable number of tasks, directly applying a model trained on approximately 200-task instances to graphs with 1000–2000 tasks introduces substantial computational and distributional shift. In particular, the dense Task–Task reasoning in CRST requires relation and attention tensors whose size grows quadratically with the number of visible tasks. We therefore introduce a bounded-context decomposition strategy for large-scale inference.

Importantly, the decomposition is applied only to the *neural observation*. The scheduling instance itself is never divided into independent subproblems. A single global event-driven environment maintains machine occupancy, reusable-resource usage, precedence states, and machine calendars throughout the complete rollout. Every selected action is committed directly to this global environment and is checked by the same hard feasibility mask as in standard CRST inference.

H.1 Overview

Let N denote the number of unfinished tasks. At each rolling step, the decomposition constructs three nested sets:

$$\mathcal{A}_t \subseteq \mathcal{H}_t \subseteq \mathcal{U}_t, \tag{85}$$

where $\mathcal{U}_t$ is the set of all unfinished tasks, $\mathcal{H}_t$ is a bounded planning context visible to the neural network, and $\mathcal{A}_t$ is the action core from which the policy is allowed to select tasks.

We use the fixed configuration

$$H = 200, \qquad A = 150, \qquad S = 50, \tag{86}$$

where H is the maximum number of visible tasks, A is the maximum number of actionable tasks, and S is the number of committed actions before the context is reconstructed. Thus, CRST always reasons over a task graph whose size remains close to that seen during training, regardless of the total instance size.

H.2 Global Context Selection

Before constructing each neural observation, we perform a lightweight deterministic analysis over all unfinished tasks. This analysis does not invoke CRST and does not construct the full heterogeneous attention graph. It estimates task urgency and bottleneck information using quantities such as earliest feasible timing, remaining critical-path length, machine flexibility, machine load, and reusable-resource pressure.

The planning context $\mathcal{H}_t$ contains at most H tasks and is formed from three complementary components:

$$\mathcal{H}_t = \mathcal{H}_t^{\text{frontier}} \cup \mathcal{H}_t^{\text{critical}} \cup \mathcal{H}_t^{\text{contention}}. \tag{87}$$

Approximately 60% of the context is allocated to the temporal scheduling frontier, 20% to tasks with long remaining critical paths, and 20% to tasks associated with strong machine or resource contention.

The selected context is predecessor-closed: when an unfinished task is added, its unfinished predecessors are also included whenever necessary. This keeps the local neural context structurally coherent and avoids exposing a task without the unfinished dependencies required to explain its readiness.

H.3 Action Core

Not every task visible to CRST is immediately allowed to produce an action. The first $A = 150$ tasks form the action core $\mathcal{A}_t \subseteq \mathcal{H}_t$, whereas the remaining context tasks participate only in representation learning:

$$\underbrace{\mathcal{A}_t}_{150 \text{ actionable}} \subset \underbrace{\mathcal{H}_t}_{200 \text{ visible}}. \tag{88}$$

This separation allows CRST to observe additional critical-chain and bottleneck context without permitting future-context tasks to be dispatched prematurely. The normal global feasibility mask is subsequently applied to task–machine pairs whose task belongs to $\mathcal{A}_t$. Hence, the decomposition restricts the candidate region but never overrides the original feasibility rules.

H.4 Rolling Commitment

Within one planning window, CRST repeatedly selects feasible task–machine pairs from the action core and commits them directly to the global simulator. After $S = 50$ actions have been committed, the global context is recomputed and a new bounded task graph is constructed.

When the number of unfinished tasks becomes no larger than the action-core capacity A, the remaining tasks are handled in the final window without further decomposition. In the ideal case where every intermediate context is sufficiently populated, the number of rolling windows is approximately

$$K = \left\lceil \frac{N - A}{S} \right\rceil + 1. \tag{89}$$

Algorithm 4: Bounded-Context CRST Inference

Input: Global scheduling state S; trained CRST policy π_θ
Output: Complete feasible schedule

```
Initialize one global scheduling environment;
while unfinished tasks remain do
    Compute lightweight global task statistics;
    Construct predecessor-closed context H_t, |H_t| ≤ H;
    Select action core A_t ⊆ H_t, |A_t| ≤ A;
    Set commit budget B ← min(S, |U_t|);
    for b = 1, ..., B do
        Construct the CRST observation over H_t;
        Apply the global feasibility mask to pairs from A_t;
        if no feasible pair exists in the action core then
            Advance the global environment to the next feasibility-changing event;
        else
            Select (i, m) using π_θ;
            Commit (i, m) directly to the global environment;
Validate the final global schedule;
```

For $N = 1000$ and $N = 2000$, this gives 18 and 38 windows, respectively, which matches the observed average number of windows in our experiments.

Unlike look-ahead schemes that construct tentative local schedules and later replay them, our procedure directly commits each selected action:

$$N_{\text{tentative}} = N_{\text{committed}} = N. \tag{90}$$

There is therefore no additional reconciliation stage between independently constructed local schedules.

H.5 Preservation of Global Constraints

The key distinction between the proposed method and conventional problem decomposition is that only the neural Task graph is bounded. All constraint states remain global across windows.

For machine capacity, every committed action updates the same global machine-availability state. Machine-rest intervals are evaluated against the same absolute calendar regardless of the current neural window. Reusable resource units similarly retain their global occupation and release times. Precedence constraints use the actual completion times of previously committed predecessors when determining successor readiness. Machine compatibility is also rechecked by the global task–machine feasibility mask for every action.

Consequently, switching from one planning context to another does not reset or approximate any hard scheduling constraint. The final result is a single continuous schedule generated by one global simulator rather than a collection of independently optimized local schedules.

H.6 Computational Complexity

The main computational motivation is the dense Task–Task component of CRST. With N visible tasks, a full forward pass contains attention and relation structures of order $O(N^2)$.Since a constructive rollout requires on the order of N task assignments, the corresponding dense component over a complete rollout can approach $O(N^3)$in the full-graph implementation.

Under bounded-context inference, at most H tasks participate in each neural evaluation. The corresponding dense interaction cost becomes $O(H^2)$per evaluation and approximately $O(NH^2)$ over a complete rollout. Since H is fixed independently of N, the quadratic neural state size no longer grows with the total task count.

For the fixed $H = 200$ configuration, increasing an instance from 200 to 2000 tasks therefore does not induce the $100\times$ growth in the dense Task–Task representation that would occur under full-graph inference.

H.7 Full-Graph Comparison

The full and decomposed variants use the same trained CRST checkpoint, global environment, hard feasibility rules, greedy decoding policy, and evaluation instances. They differ only in the Task context presented to the neural network and the range of tasks exposed to the decoder.

Method	Visible tasks	Actionable tasks	Rolling commit
Full CRST	all N	all feasible tasks	No
Decomposed CRST	≤ 200	≤ 150	50 actions

On 2000-task instances, this decomposition reduces PyTorch peak allocated GPU memory from 691.50 MB to 38.77 MB, i.e., to only 5.61% of the full-graph level. The average inference time decreases from 53.41 s to 40.74 s, while the paired average makespan increases by only 0.477%. All 60 decomposed solutions satisfy the global feasibility checks.

H.8 Limitations

The decomposition introduces a deliberate information bottleneck. CRST can only perform neural interaction reasoning among the H tasks visible in the current context; higher-order interactions involving tasks outside the window are represented only indirectly through the lightweight global statistics. Moreover, because only the first A visible tasks belong to the action core, there can be states in which no action is currently legal inside the core even though another visible task outside the core could be dispatched. In such cases the fixed strategy advances the simulation to the next event.

Finally, each group of S committed actions is irreversible. More frequent context reconstruction could provide additional opportunities for replanning but also increases inference cost and does not necessarily improve schedule quality. We therefore treat bounded-context decomposition primarily as a scalability mechanism that trades a small amount of solution quality for substantially reduced memory and computational cost, rather than as a method intended to improve the full-graph solution itself.

I Heterogeneous Graph Encoder Comparison

We further investigate whether the performance of CRST mainly arises from using a heterogeneous graph representation, or from its scheduling-specific relational design. To this end, we conduct a controlled encoder-replacement experiment in which only the three-layer graph encoder is changed, while the remaining scheduling pipeline is kept fixed. Specifically, all variants use the same task, machine, and ResourcePool features, the same task–machine action space, feasibility mask, Pair Decoder, critic, PPO training procedure, training-instance generator, and random seed. Each replacement encoder is trained from scratch without warm starting. The evaluation benchmark contains four 200-task scenarios with 30 instances per scenario, for a total of 120 instances. Thus, the comparison isolates differences in heterogeneous state representation rather than changes in the scheduling environment or policy head.

All variants operate on the same heterogeneous scheduling graph with three node types: tasks, machines, and ResourcePools. Task–Task relations describe workflow dependencies, Task–Machine edges represent processing compatibility and execution information, and Task–Resource edges represent reusable-resource requirements. Each encoder produces task, machine, and resource embeddings H_T, H_M, and H_R, which are subsequently processed by the same CRST readout. Type-specific pooling forms the global representation g_t, and the same Pair Decoder scores each feasible task–machine pair using the task and machine embeddings, their pairwise interaction, Task–Machine edge features, action-conditioned resource descriptors, and the global state. Infeasible actions are

excluded by the same hard feasibility mask. Hence, the experiment keeps the complete decision mechanism unchanged and replaces only the graph encoder.

CRST. The CRST encoder explicitly preserves three relation-specific reasoning channels. Workflow reasoning operates over the Task–Task structure, while Task–Machine and Task–Resource interactions are modeled through edge-conditioned bidirectional attention. For every task, the resulting workflow, machine, and resource contexts are kept separate before being combined through task-adaptive *Relation Mix*. Machine and ResourcePool representations are simultaneously updated through reverse-direction messages, followed by residual connections, LayerNorm, and type-specific feed-forward layers.

R-GCN. We implement an R-GCN-style encoder following relation-specific message passing (Schlichtkrull et al., 2018). For the classical variant, messages are aggregated only along explicit directed scheduling edges, including precedence relations and Task–Machine / Task–Resource interactions. Each relation transformation is represented using four shared basis matrices,

$$W_r = \sum_{k=1}^{4} a_{rk} B_k, \tag{91}$$

and the node update follows

$$h_i' = W_0 h_i + \sum_r \frac{1}{|\mathcal{N}_r(i)|} \sum_{j \in \mathcal{N}_r(i)} W_r h_j. \tag{92}$$

The encoder itself does not directly use graph distance, process scope, lag, or continuous Task–Machine / Task–Resource attributes, although these assignment-specific features remain available to the shared Pair Decoder. Residual connections, LayerNorm, and feed-forward layers are retained, and the depth and hidden dimension are matched to CRST. We also consider an enhanced scheduling-adapted R-GCN. This variant retains relation-specific transformations but additionally propagates information between unfinished tasks using multiple Task–Task relation labels and incorporates scheduling-specific structural and continuous edge information, including graph distance, process scope, processing time, ECT, and resource features. It therefore provides a stronger R-GCN-style baseline with information access closer to CRST, rather than a strict reproduction of the original sparse R-GCN.

HAN-style. The HAN-style encoder is inspired by the hierarchical attention mechanism of HAN (Wang et al., 2019). Multi-head node-level attention is first applied separately to workflow, machine-to-task, and resource-to-task views, producing three task contexts c_i^D, c_i^M, and c_i^R. Semantic-level attention then computes a graph-level mixture,

$$\boldsymbol{\alpha} = \mathrm{softmax}\left(\mathrm{Mean}_{i \in \mathcal{T}_{\mathrm{unfinished}}} \mathrm{score}(c_i^D, c_i^M, c_i^R)\right), \tag{93}$$

and updates the task representation as

$$h_i' = \mathrm{LN}\left(h_i + \sum_{r \in \{D,M,R\}} \alpha_r c_i^r\right). \tag{94}$$

Machine and ResourcePool nodes are updated through the corresponding reverse attention. Unlike the original HAN, we do not explicitly construct meta-path neighborhoods; instead, the three direct scheduling relations serve as the semantic views. We therefore refer to this implementation as *HAN-style*. Importantly, its semantic coefficients are shared across tasks within the same graph, whereas CRST computes Relation Mix weights independently for each task and encoder layer.

HGT-style. The HGT-style encoder follows the type- and relation-dependent attention principle of HGT (Hu et al., 2020). Task, machine, and ResourcePool nodes use type-specific query, key, and value projections, while directed relation types use relation-specific key/value transformations and attention priors. Heterogeneous incoming messages are jointly normalized at each target node, continuous edge attributes provide additional attention biases, and each node type uses its own output projection and learnable residual strength. The implementation captures the central heterogeneous attention mechanism of HGT, but does not reproduce HGSampling or the complete relative temporal encoding of the original model. The scheduling graph is instead processed using the same batching and masking framework as CRST; hence we denote it *HGT-style*.

Results. Table 5 shows that CRST achieves the lowest greedy makespan across all four benchmark scenarios under the common training budget. Among the alternative encoders, the enhanced R-GCN is closest to CRST, while the HGT-style encoder shows the largest overall degradation. The difference is relatively small on Resource Scale but becomes more pronounced on Cross-Process, Hub Flexibility, and Machine Multi-Mode, where multiple interaction mechanisms must be considered simultaneously.

We additionally evaluate sample-64 decoding. Sampling improves the overall mean makespan of all compared encoders: CRST improves by 2.90%, the enhanced R-GCN by 2.64%, HAN-style by 0.59%, and HGT-style by 4.87%. Nevertheless, CRST retains the best overall solution quality after sampling, with an average makespan of 2541.6 compared with 2588.1, 2661.6, and 2644.0 for the enhanced R-GCN, HAN-style, and HGT-style variants, respectively.

These comparisons suggest that the benefit of CRST cannot be explained solely by representing the scheduling problem as a heterogeneous graph. Generic relation-aware message passing or heterogeneous attention can exploit part of the same structural information, but CRST explicitly separates workflow, machine, and reusable-resource reasoning before performing task-adaptive composition. The results are therefore consistent with the motivation behind Relation Mix: different tasks may require different combinations of relational information at the same scheduling state.

Limitations. This experiment is intended as a controlled encoder comparison rather than a general ranking of R-GCN, HAN, HGT, and CRST. The architectures have different parameter counts and are evaluated under a common training budget instead of independently optimized hyperparameter budgets. In addition, the HAN-style and HGT-style models are scheduling-oriented adaptations of the original architectures rather than exact reproductions, and the reported comparison is based on a single training seed. Consequently, the results support the effectiveness of CRST's scheduling-specific relational decomposition within the controlled pipeline considered here, but do not imply that the original R-GCN, HAN, or HGT architectures are universally inferior on other tasks or under different optimization settings.

I.1 Optimization Settings

Unless otherwise stated, all experiments use the same PPO configuration shown in Table 6.

Table 6: Training hyperparameters used for CRST.

Hyperparameter	Value
Discount factor γ	1.0
GAE parameter λ	0.95
PPO clipping parameter ϵ	0.20
PPO epochs per update	2
Minibatch size	16
Learning rate	1×10^{-4}
Weight decay	1×10^{-5}
Value-loss coefficient λ_V	0.50
Entropy coefficient λ_H	0.01
Auxiliary-loss coefficient λ_{aux}	0.02
Potential-shaping coefficient β	0.10
Gradient-norm clipping	1.0