SOLVING IN IMAGINATION: MULTI-STEP LATENT PLANNING FOR COMBINATORIAL SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural constructive policies have attracted increasing attention for combinatorial scheduling by learning to construct schedules through sequential decisions. However, existing methods primarily optimize representations for immediate action selection and recompute them by re-encoding the exact state after each action rather than explicitly modeling how decision-relevant representations evolve across successive scheduling steps. Moreover, repeatedly re-encoding the evolving partial schedule introduces substantial redundant computation during inference. To address these limitations, we introduce the Multi-Step Latent Planning Model (MSLPM) for constructive combinatorial scheduling. We develop a solve-then-imagine training scheme that first jointly trains an Encoder and Actor to establish a decision-relevant latent space, and then freezes both modules while learning Dynamics to propagate this fixed representation under scheduling actions through latent alignment and decision-consistency supervision. Building on these learned latent dynamics, we introduce a multi-step latent planning mechanism that recursively generates an H-step action prefix from a single exact-state encoding by alternating the Actor and Dynamics, while exact scheduling operators maintain feasibility. Across three benchmarks, i.e., PFSP, JSSP, and FJSP, MSLPM achieves lower average gaps than the greedy neural baselines considered, while multi-step latent planning substantially reduces inference time by replacing intermediate re-encoding with latent transitions. The code will be publicly released upon acceptance.

1 INTRODUCTION

Combinatorial scheduling requires a sequence of interdependent decisions under precedence, resource, and other problem-specific constraints (Bengio et al., 2021; Cappart et al., 2023). A decision made at one construction step not only determines the current partial schedule, but also changes the structure and opportunities available to subsequent decisions. Schedule construction is therefore inherently sequential: effective solvers must maintain information that remains useful as the partial solution evolves, rather than treating each decision in isolation. At the same time, practical neural schedulers must make these successive decisions efficiently, since large instances may require hundreds or thousands of construction steps. The central challenge is thus to maintain decision-relevant information across successive scheduling steps while avoiding unnecessary computation.

Neural constructive schedulers typically encode the current partial schedule and select the next action from the resulting representation (Zhang et al., 2020; Park et al., 2021; Xiao et al., 2026). Although effective, this procedure repeatedly reconstructs a representation from the current exact state, even though successive states differ only by the most recently selected action. Latent world models learn action-conditioned transitions between compact internal states, from early recurrent models and Dreamer to recent scalable and pretrained-feature models (Ha & Schmidhuber, 2018; Hafner et al., 2020; 2025; Zhou et al., 2025). Applying this idea to combinatorial scheduling, however, requires more than accurate one-step latent prediction. The predicted representation must preserve the information used by the scheduling policy, since small errors can alter action rankings and compound over recursive transitions. Meanwhile, feasibility cannot depend on prediction accuracy: precedence, resource constraints, and admissible actions must remain exact throughout schedule construction. These requirements motivate a scheduling world model that can propagate

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.

policy-relevant representations across multiple decisions while retaining an exact interface for feasibility.

We introduce the Multi-Step Latent Planning Model (MSLPM) for constructive combinatorial scheduling, built on a solve-then-imagine principle. We first train an Encoder and Actor jointly on exact scheduling decisions, allowing the policy itself to define a decision-relevant latent space. We then freeze both modules and train Dynamics to model how this representation evolves under scheduling actions through paired exact and imagined planning, supervised by latent alignment and decision consistency. At inference, MSLPM alternates frozen Actor and learned Dynamics to generate an action prefix of up to H steps from a single exact-state encoding, replacing intermediate Encoder evaluations with latent transitions. Exact scheduling operators maintain feasible action masks and symbolic state updates, while periodic re-encoding limits accumulated prediction drift. We instantiate the same architecture and training strategy on PFSP, JSSP, and FJSP to evaluate the framework across different scheduling constraints.

Our contributions are threefold:

- We introduce MSLPM for constructive combinatorial scheduling. Its solve-then-imagine strategy first learns a decision-relevant policy representation and then trains Dynamics to predict how this representation evolves under scheduling actions using multi-step latent and decision supervision.
- Using this learned world model, we develop a multi-step latent planning mechanism that generates an H-step action prefix from a single exact-state encoding by alternating the Actor and Dynamics. Replacing intermediate re-encoding with latent transitions reduces inference cost during schedule construction.
- We validate the same architecture and training strategy on PFSP, JSSP, and FJSP. MSLPM improves solution quality over the compared greedy neural solvers, while multi-step latent planning accelerates inference relative to direct policy decoding.

2 Related Work

Neural constructive scheduling and generalization. Zhang et al. (2020) and Park et al. (2021) formulated scheduling as sequential decision-making over partial schedules and used graph-based policies to model operation dependencies and scheduling states. Wang et al. (2024) further introduced attention mechanisms to capture richer operation–machine interactions, while Xiao et al. (2026) simplified state modeling with a Transformer-based scheduling architecture. Beyond these graph- and attention-based approaches, Li et al. (2024), Corsini et al. (2024), Zhao et al. (2022), and Luttmann & Xie (2026) explored imitation learning, reinforcement learning, self-improvement, and multi-action decision-making across different scheduling settings. These developments improve state-aware decision-making, but also expose a computational tension: Kool et al. (2019) showed that encoding an instance once enables efficient decoding, whereas state-aware scheduling methods commonly re-encode the evolving partial solution to refresh its representation. More recently, Drakulić et al. (2025) proposed GOAL to share architectures across multiple combinatorial problems, and Pan et al. (2025) introduced UniCO to unify several problem classes through a common reduced formulation. In contrast to universal or cross-task solvers, our work studies a common latent-planning principle across scheduling families and propagates the policy's decision state between periodic exact encodings.

Decision-oriented latent world models. Hafner et al. (2019) introduced PlaNet, which learns latent dynamics for planning directly in a compact state space. Hafner et al. (2020) extended this idea in Dreamer by learning behavior from imagined latent trajectories, while Schrittwieser et al. (2020) showed with MuZero that useful dynamics can be learned without reconstructing observations. Schwarzer et al. (2021) and Ni et al. (2024) further investigated self-predictive representations and the latent abstractions required for future-state prediction. More recent work has explored discrete latent states, pretrained feature prediction, and efficient imagined rollouts: Scannell et al. (2025) developed discrete codebook world models, Zhou et al. (2025) predicted pretrained visual features for planning, and Chun et al. (2026) reduced the representation cost of imagined rollouts. These methods typically learn representations jointly with dynamics or control, or rely on representations obtained from separate predictive objectives. Our approach instead first trains the scheduling

policy to define a decision-relevant latent space, and then freezes the Encoder and Actor before learning Dynamics to preserve both successor representations and the Actor's decision behavior.

Planning and inference-time computation. Pirnay & Grimm (2024) and Hottung et al. (2025) improved constructive optimization by evaluating multiple trajectories or diverse solution policies at inference time. Zhang et al. (2024) introduced a learned improvement heuristic that searches over complete solutions, while Chen et al. (2026) proposed MaskCO to iteratively regenerate selected solution components. Li et al. (2025) further incorporated learning into rolling-horizon optimization by repeatedly solving overlapping subproblems. Although these methods improve solution quality through additional inference-time computation, they generally rely on exploring alternatives, revising solutions, or repeatedly solving explicit subproblems. Our use of planning follows a different direction: instead of increasing the search budget, we reuse a learned decision representation across successive actions. Starting from one exact encoding, the frozen Actor and learned Dynamics recursively generate a greedy H-step action prefix, while exact masks preserve feasibility and periodic re-encoding limits prediction drift. The planning horizon therefore controls the quality–runtime trade-off through latent reuse rather than additional search.

3 PRELIMINARIES

3.1 COMBINATORIAL SCHEDULING

Let $x = (\mathcal{V}_x, p_x, \Gamma_x)$ denote a combinatorial scheduling instance, where $\mathcal{V}_x$ is a finite set of activities. $p_x = (p_i)_{i\in\mathcal{V}_x}$ contains their processing times. A schedule $S = (S_i)_{i\in\mathcal{V}_x}$ assigns a start time S_i to each activity. The completion time of S_i is $C_i := S_i + p_i$. The makespan of S is

$$C_{\max}(S) := \max_{i\in\mathcal{V}_x} C_i.$$

Problem-specific constraints are represented by the feasibility indicator Γ_x, where $\Gamma_x(S) = 1$ if and only if S is feasible for instance x. The scheduling problem is

$$min \quad C_{\max}(S) \qquad \text{subject to} \quad \Gamma_x(S) = 1. \tag{1}$$

3.2 SEQUENTIAL CONSTRUCTIVE FORMULATION

Let $N_x := |\mathcal{V}_x|$ and let $\mathcal{Q}_x$ denote the set of exact partial-schedule states. Consider a reference state $s_t \in \mathcal{Q}_x$. After selecting h actions from this state, let $\boldsymbol{a}_{t:t+h-1} := (a_t, \ldots, a_{t+h-1})$ denote the resulting action prefix. The prefix is empty when $h = 0$. Function $\mathcal{M}_x$ determines which activities can be selected after this prefix:

$$m_{t+h} := \mathcal{M}_x\left(s_t, \boldsymbol{a}_{t:t+h-1}\right) \in \{0,1\}^{N_x}. \tag{2}$$

For a fixed ordering of $\mathcal{V}_x$, the corresponding selectable action set is

$$\mathcal{A}_x(m_{t+h}) := \left\{i \in \mathcal{V}_x \mid [m_{t+h}]_i = 1\right\}. \tag{3}$$

Thus, $\mathcal{M}_x$ computes the exact action mask from the reference state and the selected prefix, while $\mathcal{A}_x$ reads the selectable activities from that mask. Selecting $a_{t+h} \in \mathcal{A}_x(m_{t+h})$ updates the corresponding exact state through the deterministic scheduling transition

$$s_{t+h+1} = \mathcal{T}_x\left(s_{t+h}, a_{t+h}\right). \tag{4}$$

Here $\mathcal{T}_x$ commits the selected activity according to the problem-specific scheduling rules. Starting from the empty state s_x^0, repeated application of the mask and transition functions produces a complete action sequence $\boldsymbol{a} = (a_0, \ldots, a_{N_x-1})$. Let Π_x denote the set of complete sequences whose actions satisfy $a_t \in \mathcal{A}_x(m_t)$ at every step. Each $\boldsymbol{a} \in \Pi_x$ produces a unique terminal schedule $S(\boldsymbol{a})$ satisfying $\Gamma_x\left(S(\boldsymbol{a})\right) = 1$. The scheduling problem is therefore expressed in the constructive decision space as

$$\boldsymbol{a}^\star \in \arg\min_{\boldsymbol{a}\in\Pi_x} C_{\max}\left(S(\boldsymbol{a})\right), \qquad S^\star = S(\boldsymbol{a}^\star). \tag{5}$$

We study three classical combinatorial scheduling problems: PFSP, JSSP and FJSP. Their constraints and constructive interfaces are detailed in Appendix A. The proof of the equivalence in Equation 5 is provided in Appendix B.

4 METHOD

4.1 OVERVIEW

Our *Multi-Step Latent Planning Model* (MSLPM), as shown in Figure 1, is a latent world model for constructive combinatorial scheduling. Given an exact state s_t, Encoder E_ϕ represents it in latent space as z_t. Actor π_θ selects an action a_t conditioned on z_t, and Dynamics F_ψ predicts the next latent representation $\widehat{z}_{t+1}$. By alternating π_θ with F_ψ, MSLPM can generate multi-step action prefixes by recursively imagining future latent states from a single encoding of s_t. This mechanism avoids repeatedly applying $\mathcal{T}_x$ and re-encoding intermediate states with E_ϕ within each action prefix. Training follows a solve-then-imagine strategy. Stage 1 learns the Encoder E_ϕ and Actor π_θ from exact scheduling decisions. Stage 2 freezes the two modules learned in Stage 1 and trains F_ψ through multi-step paired rollouts.

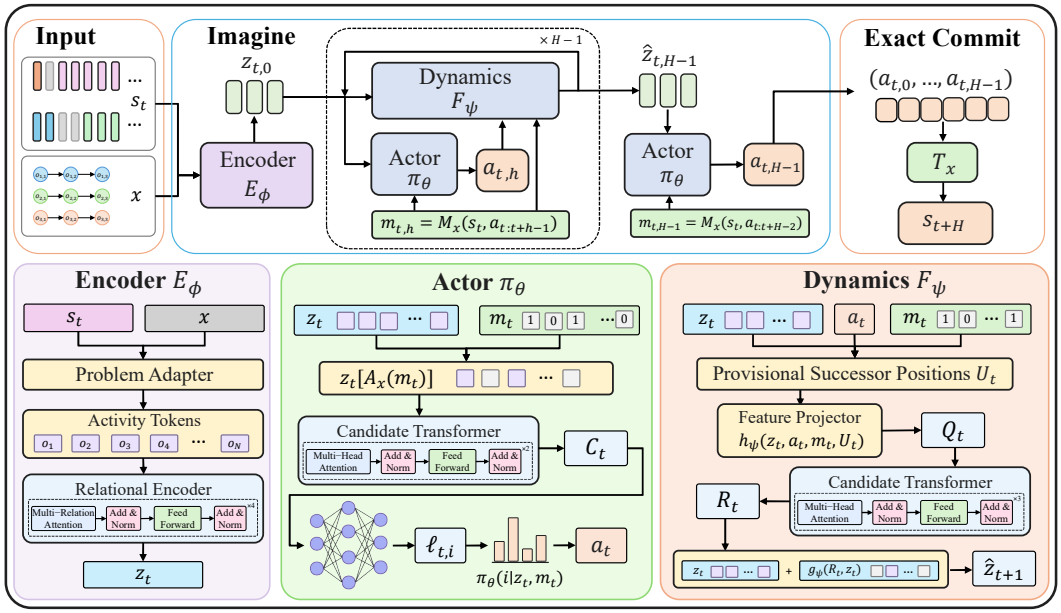

Figure 1: **Multi-step latent planning with MSLPM. Top:** Actor and Dynamics generate an H-step action prefix from one exact-state encoding. **Bottom:** Architectures of Encoder, Actor, and Dynamics.

4.2 MSLPM ARCHITECTURE

As illustrated in Figure 1, MSLPM comprises three components: an Encoder E_ϕ that maps exact states into a shared latent space, an Actor π_θ that selects feasible activities, and a Dynamics F_ψ that predicts latent transitions.

Encoder. Given an instance x and its exact state s_t, Encoder E_ϕ produces

$$z_t := E_\phi(s_t) \in \mathbb{R}^{N_x \times d}, \tag{6}$$

where $[z_t]_i$ is the d-dimensional representation of activity $i \in \mathcal{V}_x$. A problem adapter converts problem-specific state features and relations of each scheduling problem into this shared representation.

Actor. Given activity representations $z_t \in \mathbb{R}^{N_x \times d}$ and exact feasibility mask m_t, the Actor π_θ defines a distribution over selectable activities and chooses

$$a_t = \underset{i \in \mathcal{A}_x(m_t)}{\arg\max}\, \pi_\theta(i \mid z_t, m_t). \tag{7}$$

Let $n_t := |\mathcal{A}_x(m_t)|$ denote the number of selectable actions. Let $z_t[\mathcal{A}_x(m_t)] \in \mathbb{R}^{n_t \times d}$ collect corresponding latent tokens. A two-layer Candidate Transformer f_θ^{cand}, built from standard Transformer encoder blocks (Vaswani et al., 2017), contextualizes these tokens with respect to the current candidate set $\mathcal{A}_x(m_t)$:

$$C_t = f_\theta^{\text{cand}}\left(z_t[\mathcal{A}_x(m_t)]\right) \in \mathbb{R}^{n_t \times d}. \tag{8}$$

Each row of candidate token C_t represents its associated activity in the context of all other currently selectable activities $\mathcal{A}_x(m_t)$, providing the contextualized representation used for action scoring. A shared MLP g_θ then maps each candidate token C_t to a scalar logit ℓ_t for each selectable activity:

$$(\ell_{t,i})_{i \in \mathcal{A}_x(m_t)} = g_\theta(C_t) \in \mathbb{R}^{n_t}. \tag{9}$$

The Actor normalizes these logits ℓ_t over selectable activities $\mathcal{A}_x(m_t)$:

$$\pi_\theta(i \mid z_t, m_t) = \begin{cases} \dfrac{\exp(\ell_{t,i})}{\sum_{j \in \mathcal{A}_x(m_t)} \exp(\ell_{t,j})}, & i \in \mathcal{A}_x(m_t), \\ 0, & i \notin \mathcal{A}_x(m_t). \end{cases} \tag{10}$$

Dynamics. Given activity representations z_t, selected action a_t, and exact feasibility mask m_t, the Dynamics model F_ψ predicts

$$\widehat{z}_{t+1} = F_\psi(z_t, a_t, m_t) \in \mathbb{R}^{N_x \times d}. \tag{11}$$

Dynamics updates a structurally determined subset of activity tokens. Let

$$\mathcal{U}_t = \{u_{t,1}, \ldots, u_{t,q_t}\} \subseteq \mathcal{V}_x, \qquad q_t := |\mathcal{U}_t|, \tag{12}$$

denote the provisional successor positions determined from m_t and a_t. Their ordering follows the fixed ordering of $\mathcal{V}_x$. The set $\mathcal{U}_t$ specifies where predicted latent residuals are placed. The exact next-step action set remains determined by $\mathcal{M}_x$.

For each $u_{t,r} \in \mathcal{U}_t$, the feature projector h_ψ combines its existing activity token with the current token from the same structural slot, the selected action token $[z_t]_{a_t}$, pooled candidate context, and structural progress features. The resulting transition-token matrix is

$$Q_t = h_\psi(z_t, a_t, m_t, \mathcal{U}_t) \in \mathbb{R}^{q_t \times d}. \tag{13}$$

A three-layer Candidate Transition Transformer f_ψ^{trans} contextualizes transition tokens Q_t through self-attention:

$$R_t = f_\psi^{\text{trans}}(Q_t) \in \mathbb{R}^{q_t \times d}. \tag{14}$$

Output R_t contains contextualized transition features. A residual head g_ψ combines R_t with corresponding base activity tokens in z_t to predict candidate-level residuals using a shared MLP. Predicted residuals update corresponding activity tokens, while all other tokens in z_t remain unchanged. Dynamics F_ψ then updates

$$\widehat{z}_{t+1} = z_t + g_\psi(R_t, z_t). \tag{15}$$

Although it uses problem-specific adapters, MSLPM retains the same Encoder–Actor–Dynamics architecture and training procedure across scheduling problem classes. A separate model is trained for each problem class, with a single set of parameters shared across all instance sizes within that class.

4.3 Multi-Step Latent Planning

By predicting latent state transitions, Dynamics enables MSLPM to perform multi-step latent planning. Figure 1 illustrates this procedure. Given an exact root state s_t, MSLPM computes $z_t = E_\phi(s_t)$ and initializes $\widehat{z}_t = z_t$. Define $L_t := \min\{H, N_x - t\}$ as the current prefix length. MSLPM then generates L_t actions without re-encoding intermediate states. For $h = 0, \ldots, L_t - 1$, it recursively computes

$$\begin{aligned} m_{t+h} &= \mathcal{M}_x\left(s_t, \boldsymbol{a}_{t:t+h-1}\right), \\ a_{t+h} &= \underset{i \in \mathcal{A}_x(m_{t+h})}{\arg\max}\ \pi_\theta\left(i \mid \widehat{z}_{t+h}, m_{t+h}\right), \\ \widehat{z}_{t+h+1} &= F_\psi\left(\widehat{z}_{t+h}, a_{t+h}, m_{t+h}\right). \end{aligned} \tag{16}$$

At each latent planning step, $\mathcal{M}_x$ derives an exact feasibility mask from root state s_t and current action prefix. Actor π_θ selects a feasible action under this mask, while Dynamics F_ψ predicts a successor latent representation. After generating a prefix, MSLPM commits each action using exact transition function $\mathcal{T}_x$:

$$s_{t+h+1} = \mathcal{T}_x(s_{t+h}, a_{t+h}), \qquad h = 0, \ldots, L_t - 1. \tag{17}$$

State s_{t+L_t} then becomes the root of the next planning cycle. Planning repeats until schedule construction is complete.

With $H = 1$, MSLPM reduces to direct Encoder–Actor decoding and re-encodes each exact successor state. With $H > 1$, Dynamics replaces intermediate Encoder evaluations within each action prefix. Constructing a schedule with N_x decisions therefore requires $\lceil N_x/H \rceil$ Encoder evaluations rather than N_x.

4.4 Learning to Solve Before Learning to Imagine

Existing world models either learn predictive representations jointly from observations or model dynamics in pretrained feature spaces (Hafner et al., 2025; Bardes et al., 2024; Zhou et al., 2025). MSLPM follows a two-stage solve-then-imagine training strategy. **Learning to solve.** Stage 1 learns a latent representation that supports scheduling decisions. **Learning to imagine.** Stage 2 freezes parameters of E_ϕ and π_θ, and trains F_ψ to predict action-conditioned latent transitions over multi-step imagined trajectories.

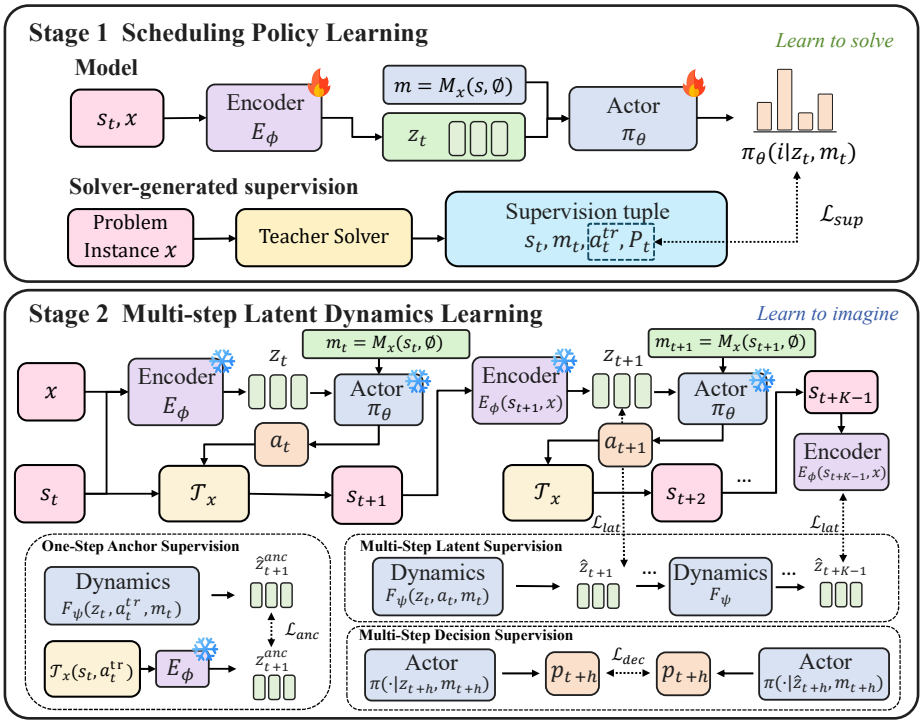

Figure 2: **Solve-then-imagine training of MSLPM. Stage 1:** The Encoder and Actor are jointly trained to establish a decision-relevant latent space. **Stage 2:** With both modules frozen, Dynamics is trained on paired exact and imagined rollouts using one-step anchor, multi-step latent, and decision supervision.

Stage 1: Learning to Solve. Let $\mathcal{D}_1$ consist of training tuples collected from solver-generated teacher trajectories. Each tuple contains an instance x, an exact partial-schedule state s, its mask $m = \mathcal{M}_x(s, ())$, a recorded action a^{tr}, and a nonempty set $\mathcal{P} \subseteq \mathcal{A}_x(m)$ of actions certified by the teacher trajectory, with $a^{\mathrm{tr}} \in \mathcal{P}$. For notational convenience, we write the composed Encoder–Actor

policy as $\pi_\theta(\cdot \mid E_\phi(s), m)$ and train it end to end using

$$\mathcal{L}_{\text{solve}}(\phi, \theta) := \mathbb{E}_{(s, a^{\text{tr}}, \mathcal{P}) \sim \mathcal{D}_1} \left[\mathcal{L}_{\text{sup}} \left(\pi_\theta \left(\cdot \mid E_\phi(s), m \right), a^{\text{tr}}, \mathcal{P} \right) \right]. \tag{18}$$

The supervised objective follows the recorded action while avoiding arbitrary penalties on other teacher-certified actions. Consequently, E_ϕ learns a policy-induced latent representation optimized for scheduling decisions rather than state reconstruction. The complete supervised objective is provided in Appendix C.1.

Stage 2: Learning to Imagine. We freeze E_ϕ and π_θ and train F_ψ using a dataset $\mathcal{D}_2$ of K-step rollout windows. For each window, we pair an exact branch with an imagined branch initialized from root encoding z_t. Both branches follow the same feasible actions a_{t+h} under exact masks m_{t+h}. Exact branch re-encodes each successor to provide supervision, while imagined branch recursively predicts from its own previous outputs. For $h = 0, \ldots, K-1$, the paired rollouts satisfy

$$\begin{aligned} s_{t+h+1} &= \mathcal{T}_x(s_{t+h}, a_{t+h}), \\ z_{t+h+1} &= E_\phi(s_{t+h+1}), \\ \widehat{z}_{t+h+1} &= F_\psi(\widehat{z}_{t+h}, a_{t+h}, m_{t+h}). \end{aligned} \tag{19}$$

During training, action selection gradually shifts from recorded replay actions toward greedy actions selected by the frozen Actor from imagined representations. After the imagined branch deviates from the recorded prefix, the exact branch follows the same selected actions. Its successor encodings therefore remain matched to the imagined trajectory. We train F_ψ using multi-step latent and decision supervision, together with a teacher-forced one-step anchor.

Latent alignment loss $\mathcal{L}_{\text{lat}}(\widehat{z}_{t+h}, z_{t+h}, m_{t+h})$ aligns predicted activity tokens $\widehat{z}_{t+h}$ with exact successor encodings z_{t+h} across rollout depths h (Schwarzer et al., 2021; Ni et al., 2024).

Decision consistency loss $\mathcal{L}_{\text{dec}}(\widehat{z}_{t+h}, z_{t+h}, m_{t+h})$ compares their consequences for action selection. Frozen Actor produces $\pi_\theta(\cdot \mid \widehat{z}_{t+h}, m_{t+h})$ and $\pi_\theta(\cdot \mid z_{t+h}, m_{t+h})$. KL divergence matches the imagined distribution to the exact distribution, following the policy-distillation principle (Rusu et al., 2016; Voelcker et al., 2025). The exact Actor's greedy action additionally supplies a cross-entropy target. A ranking-margin term encourages this action to outrank competing candidates.

One-step anchor $\mathcal{L}_{\text{anc}}$ compares the prediction $F_\psi(z_t, a_t^{\text{tr}}, m_t)$ with exact target $E_\phi(\mathcal{T}_x(s_t, a_t^{\text{tr}}))$. a_t^{tr} denotes the recorded replay action. It supervises the successor representations and their latent changes relative to z_t. It also applies decision consistency under corresponding exact successor mask. This retains direct supervision from freshly encoded states, while the multi-step branch trains recursive predictions from imagined representations.

These terms form the weighted objective

$$\begin{aligned} \mathcal{L}_{\text{imagine}}(\psi) := \mathbb{E}_{\mathcal{D}_2} \Bigg[\sum_{h=1}^{K} w_{K,h} \Big(& \lambda_{\text{lat}} \mathcal{L}_{\text{lat}} \left(\widehat{z}_{t+h}, z_{t+h}, m_{t+h} \right) + \mathcal{L}_{\text{dec}} \left(\widehat{z}_{t+h}, z_{t+h}, m_{t+h} \right) \Big) \\ & + \lambda_{\text{anc}} \mathcal{L}_{\text{anc}} \left(s_t, a_t^{\text{tr}}, m_t \right) \Bigg]. \end{aligned} \tag{20}$$

Gradients propagate through imagined rollout to update only F_ψ. Encoder and Actor parameters remain unchanged. Complete loss definitions and weighting settings are provided in Appendix C.2.

5 Experiments

5.1 Experimental Setup

Benchmarks and Baselines. We evaluate MSLPM on PFSP, JSSP, and FJSP. For PFSP, we evaluate Taillard (Taillard, 1993). For JSSP, we evaluate Taillard, Demirkol, and Lawrence (Taillard, 1993; Demirkol et al., 1998; Lawrence, 1984). For FJSP, we use Brandimarte and Hurink's edata, rdata, and vdata benchmarks (Brandimarte, 1993; Hurink et al., 1994). We compare three types of baselines. **Rule-based methods** include SPT and LPT for PFSP; LPT and MOR for JSSP;

and FIFO-SPT, MOR-EET, and MWKR-SPT for FJSP. **Greedy learning-based methods** include Graph-IL (Li et al., 2024) and PFSPNet (Pan et al., 2023) for PFSP; L2D (Zhang et al., 2020), SLIM (Corsini et al., 2024), and ReSCHED (Xiao et al., 2026) for JSSP; and DANIEL-G (Wang et al., 2024), DOAGNN-G (Zhao et al., 2025), and ReSCHED for FJSP. **Solver-based methods** include OR-Tools CP-SAT (Perron et al., 2023) for all three problems.

Training and Evaluation Settings. For each problem, we train a separate model using 1,000–2,000 generated instances and select checkpoints on a separately generated validation set. For JSSP and PFSP, the reference makespans are the published best-known solution (BKS) values provided by the benchmark suites. For FJSP, the best feasible makespan found by CP-SAT (Perron et al., 2023) using 8 workers and a time limit of 1,800 seconds per instance is used as the reference makespan. For instance i, the optimality gap is computed as

$$\mathrm{Gap}_i(\%) = 100 \times \frac{C_i - C_i^{\mathrm{ref}}}{C_i^{\mathrm{ref}}},$$

where C_i denotes the makespan produced by the evaluated method and C_i^{ref} denotes the corresponding reference makespan. All runtimes are measured on the same hardware. Further details are provided in Appendix D.

5.2 Main Results

Tables 1, 2, and 3 report main results on PFSP, JSSP and FJSP, respectively. Detailed FJSP and JSSP results are provided in Appendix E. At $H = 1$, MSLPM makes each scheduling decision from an exact Encoder representation, without invoking Dynamics. Compared with the strongest greedy learning-based baseline, the average gap decreases from 9.01% to 8.44% on FJSP, from 16.37% to 12.14% on JSSP, and from 23.35% to 11.08% on PFSP. As the multi-step latent planning horizon H increases, Dynamics replaces intermediate exact encodings with recursive latent transitions, reducing mean runtime from 0.20 to 0.07 seconds on FJSP, from 2.94 to 1.36 seconds on JSSP, and from 1.85 to 0.80 seconds on PFSP when H increases from 1 to 5, corresponding to $2.86\times$, $2.16\times$, and $2.31\times$ speedups, respectively. The effect of rollout depth on solution quality is problem-dependent: $H = 5$ gives the lowest average gap on FJSP, $H = 3$ on JSSP, and $H = 1$ on PFSP, while the two multi-step settings remain close on PFSP. These results show that MSLPM substantially reduces repeated re-encoding cost while maintaining competitive scheduling performance.

Table 1: **Results on the Taillard PFSP benchmarks.** Each entry reports the mean gap (%) and mean runtime (s) as Gap/Time. Each size group contains 10 instances.

Size	**Rule-based**		**Greedy learning-based**		**Solver-based**	**MSLPM (ours)**		
	SPT	**LPT**	**Graph-IL**	**PFSPNet**	**CP-SAT$_{5s}$**	$H = 1$	$H = 3$	$H = 5$
20×5	22.34/0.01	26.21/0.01	26.33/0.05	29.80/0.03	1.56/5.02	9.16/0.43	10.99/0.27	10.67/0.22
20×10	26.38/0.01	28.46/0.01	30.88/0.03	29.91/0.03	3.39/5.01	15.30/0.87	18.17/0.53	18.71/0.45
50×5	17.92/0.01	16.46/0.01	19.50/0.11	23.74/0.08	12.10/5.02	8.04/1.17	7.54/0.71	7.78/0.61
50×10	27.73/0.01	28.24/0.01	26.17/0.08	25.31/0.08	10.34/5.04	13.51/2.39	15.50/1.45	14.78/1.25
50×20	28.05/0.01	28.43/0.01	26.26/0.08	26.74/0.08	17.37/5.06	14.85/5.29	15.51/3.24	16.85/2.71
100×5	14.66/0.01	12.01/0.01	14.43/0.22	17.44/0.15	9.70/5.06	4.99/2.42	5.20/1.48	4.75/1.26
100×10	18.69/0.01	19.03/0.01	19.23/0.15	19.03/0.15	16.96/5.11	8.65/5.51	8.91/3.35	9.06/2.79
100×20	23.19/0.01	22.24/0.01	23.97/0.16	25.53/0.15	23.35/5.21	14.14/14.76	14.34/7.82	14.70/6.39
Average	22.37/0.01	22.64/0.01	23.35/0.11	24.69/0.09	11.85/5.07	11.08/1.85	12.02/0.98	12.16/0.80

5.3 Ablation Study

We analyze three components of the Stage-2 objective on FJSP. All variants start from the same Stage-1 checkpoint, keep the Encoder and Actor frozen, and retrain only Dynamics. Table 4 reports results at $H \in \{3, 5\}$, where Dynamics is applied recursively. Removing $\mathcal{L}_{\mathrm{lat}}$ produces the largest degradation, increasing the gap by 1.62 points at $H = 3$ and 4.67 points at $H = 5$. Thus, aligning recursive predictions with exact Encoder representations is the main supervision that keeps the rollout on the learned policy latent space. Removing $\mathcal{L}_{\mathrm{anc}}$ further shows the benefit of grounding Dynamics at exactly encoded states, while the degradation without $\mathcal{L}_{\mathrm{dec}}$ indicates that latent

Table 2: **Results on JSSP benchmarks.** Each entry reports the mean gap (%) and mean runtime (s) as Gap/Time. LA, TA, and DMU contain 40, 80, and 80 instances, respectively.

Dataset	Rule-based		Greedy learning-based			Solver-based	MSLPM (ours)		
	LPT	MOR	L2D	SLIM	ReSCHED	CP-SAT$_{5s}$	$H=1$	$H=3$	$H=5$
LA	33.17/0.00	13.74/0.01	28.00/0.25	8.99/0.13	12.74/1.66	0.49/2.082	6.99/1.163	6.55/0.677	7.78/0.566
TA	43.07/0.05	19.73/0.09	35.35/4.85	14.74/0.97	19.43/7.96	10.83/5.061	12.69/3.967	12.24/2.134	12.65/1.750
DMU	51.11/0.03	33.78/0.04	44.95/2.18	21.68/0.71	29.39/6.99	20.94/5.080	14.16/2.806	14.68/1.603	14.90/1.370
Overall	44.31/0.03	24.15/0.05	37.72/2.86	16.37/0.70	22.08/6.31	12.81/4.473	12.14/2.942	12.08/1.630	12.58/1.361

Table 3: **Results on FJSP benchmarks.** Each entry reports the mean gap (%) and mean runtime (s) as Gap/Time. CP-SAT$_{1,800\,s}$ defines the instance-wise reference makespan C_i^{ref} and is reported as Gap (mean reference makespan). The overall result covers all 130 instances.

Dataset	Rule-based			Greedy learning-based			Solver-based	MSLPM (ours)		
	FIFO-SPT	MOR-EET	MWKR-SPT	DANIEL-G	DOAGNN-G	ReSCHED	CP-SAT$_{1,800\,s}$	$H=1$	$H=3$	$H=5$
Brandimarte	27.99/0.12	24.36/0.12	25.16/0.12	10.35/0.89	27.26/1.00	6.65/1.42	0.00 (176.00)	5.47/0.45	8.45/0.21	6.32/0.16
Hurink (edata)	20.91/0.52	19.32/0.52	18.68/0.52	16.41/1.37	16.22/0.88	15.82/1.18	0.00 (1028.20)	13.55/0.18	14.91/0.09	13.24/0.07
Hurink (rdata)	16.27/0.52	14.11/0.53	12.92/0.52	10.51/1.37	10.77/0.88	9.74/1.20	0.00 (942.05)	9.69/0.18	10.73/0.09	9.04/0.07
Hurink (vdata)	6.21/0.52	4.34/0.52	2.89/0.52	1.97/1.37	3.01/0.89	2.07/1.21	0.00 (931.75)	2.81/0.18	2.14/0.09	1.87/0.07
Overall	15.50/0.49	13.50/0.49	12.55/0.49	9.69/1.33	11.33/0.89	9.01/1.21	0.00 (906.46)	8.44/0.20	9.20/0.09	7.92/0.07

proximity alone does not guarantee agreement with the frozen Actor. Notably, all three ablations become more detrimental at $H = 5$. Since the same Dynamics is recursively applied across multiple decisions, transition errors have more opportunity to propagate at longer horizons.

Table 4: **Stage-2 ablation on FJSP.** Mean gap (%) over all 130 Brandimarte and Hurink instances. Δ denotes the increase over the full model at the same rollout horizon.

Variant	$H=3$		$H=5$	
	Gap	Δ	Gap	Δ
MSLPM (full)	9.20	–	7.92	–
w/o Latent Alignment ($\mathcal{L}_{\mathrm{lat}}$)	10.82	+1.62	12.59	+4.67
w/o Decision Consistency ($\mathcal{L}_{\mathrm{dec}}$)	10.11	+0.91	10.57	+2.65
w/o One-Step Anchor ($\mathcal{L}_{\mathrm{anc}}$)	10.24	+1.04	10.79	+2.87

6 Conclusion

We presented MSLPM, a latent world model for constructive combinatorial scheduling that learns an Encoder–Actor solver and action-conditioned Dynamics through solve-then-imagine training. Experiments show that MSLPM consistently outperforms the strongest greedy learning-based baselines across three representative combinatorial scheduling problems. Using learned Dynamics, our multi-step latent planning mechanism replaces repeated exact-state encodings with latent transitions. Our proposed mechanism further improves inference efficiency with minor variations in solution quality but still outperforms the strongest greedy learning-based baselines. However, the relationship between rollout depth H and solution quality remains problem-dependent. Future work will investigate adaptive rollout horizons that dynamically balance solution quality and inference efficiency.

AI USE STATEMENT

Generative AI tools were used solely for language polishing. All resulting edits were reviewed by the authors.

ETHICAL STATEMENT

This research studies combinatorial scheduling using synthetic instances and established benchmarks and does not involve personal data or human subjects. Applications that affect people or resource access require additional consideration of fairness, appropriate objectives, and operational constraints. We report the method, experimental procedures, and limitations to support such assessment.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide detailed descriptions of the training objectives, data construction, evaluation protocol, and implementation settings in Appendices C and D. The complete training and evaluation code for MSLPM will be publicly released upon acceptance. We hope these resources will facilitate reproducible research and further progress in efficient neural methods for combinatorial scheduling.

REFERENCES

Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mido Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=QaCCuDfBk2.

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

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

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

Lvda Chen, Yang Li, and Junchi Yan. MaskCO: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *International Conference on Learning Representations*, 2026.

Junha Chun, Youngjoon Jeong, and Taesup Kim. Sparse imagination for efficient visual world model planning. In *International Conference on Learning Representations*, 2026.

Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3349.

Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998. doi: 10.1016/S0377-2217(97)00019-2.

Darko Drakulić, Sofia Michel, and Jean-Marc Andreoli. GOAL: A generalist combinatorial optimization agent learner. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=z2z9suDRjw.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, volume 31, pp. 2455–2467, 2018. URL https://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565, 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Danijar Hafner, Wilson Yan, and Timothy P. Lillicrap. Training agents inside of scalable world models. *arXiv preprint arXiv:2509.24527*, 2025. doi: 10.48550/arXiv.2509.24527. URL https://arxiv.org/abs/2509.24527.

André Hottung, Mridul Mahajan, and Kevin Tierney. PolyNet: Learning diverse solution strategies for neural combinatorial optimization. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=TKuYWeFE6S.

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

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1803.08475.

Stephen R. Lawrence. Resource-constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1984.

Longkang Li, Siyuan Liang, Zihao Zhu, Chris Ding, Hongyuan Zha, and Baoyuan Wu. Learning to optimize permutation flow shop scheduling via graph-based imitation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20185–20193, 2024. doi: 10.1609/aaai.v38i18.29998.

Sirui Li, Wenbin Ouyang, Yining Ma, and Cathy Wu. Learning-guided rolling horizon optimization for long-horizon flexible job-shop scheduling. In *International Conference on Learning Representations*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Laurin Luttmann and Lin Xie. Multi-action self-improvement for neural combinatorial optimization. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=6KrETIaOYD.

Tianwei Ni, Benjamin Eysenbach, Erfan Seyedsalehi, Michel Ma, Clement Gehring, Aditya Mahajan, and Pierre-Luc Bacon. Bridging state and history representations: Understanding self-predictive RL. In *International Conference on Learning Representations*, 2024.

Wenzheng Pan, Hao Xiong, Jiale Ma, Wentao Zhao, Yang Li, and Junchi Yan. UniCO: On unified combinatorial optimization via problem reduction to matrix-encoded general TSP. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yEwakMNIex.

Zixiao Pan, Ling Wang, Jingjing Wang, and Jiawen Lu. Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):983–994, 2023. doi: 10.1109/TETCI.2021.3098354.

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

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

Jonathan Pirnay and Dominik G. Grimm. Take a step and reconsider: Sequence decoding for self-improved neural combinatorial optimization. In *European Conference on Artificial Intelligence*, pp. 1927–1934, 2024. doi: 10.3233/FAIA240707.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.06295.

Aidan Scannell, Mohammadreza Nakhaeinezhadfard, Kalle Kujanpää, Yi Zhao, Kevin Sebastian Luck, Arno Solin, and Joni Pajarinen. Discrete codebook world models for continuous control. In *International Conference on Learning Representations*, 2025.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020. doi: 10.1038/s41586-020-03051-4.

Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, 2017.

Claas A. Voelcker, Anastasiia Pedan, Arash Ahmadian, Romina Abachi, Igor Gilitschenski, and Amir-Massoud Farahmand. Calibrated value-aware model learning with probabilistic environment models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61745–61768. PMLR, 2025. URL https://proceedings.mlr.press/v267/voelcker25a.html.

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.

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

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5824–5836, 2020.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1621–1632, 2020.

Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. In *International Conference on Learning Representations*, 2024.

Peng Zhao, You Zhou, Di Wang, Zhiguang Cao, Yubin Xiao, Xuan Wu, Yuanshu Li, Hongjia Liu, Wei Du, Yuan Jiang, and Liupu Wang. Dual operation aggregation graph neural networks for solving flexible job-shop scheduling problem with reinforcement learning. In *Proceedings of the ACM Web Conference 2025*, pp. 4089–4100, 2025. doi: 10.1145/3696410.3714616.

Xiaohan Zhao, Wen Song, Qiqiang Li, Huadong Shi, Zhichao Kang, and Chunmei Zhang. A deep reinforcement learning approach for resource-constrained project scheduling. In *IEEE Symposium Series on Computational Intelligence*, pp. 1226–1234, 2022. doi: 10.1109/SSCI51031.2022.10022122.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79115–79135, 2025. URL https://proceedings.mlr.press/v267/zhou25t.html.

A PROBLEM-SPECIFIC SCHEDULING INTERFACES

This appendix instantiates Γ_x, $\mathcal{M}_x$, and $\mathcal{T}_x$ for the three scheduling problems studied in this work. We consider the standard makespan variants with known processing times and non-preemptive execution. Real activities have $p_i > 0$. No additional release-time, setup-time, or machine-availability constraints are imposed. PFSP permits waiting between machines, while FJSP allows each operation to be assigned to one of its eligible machines with machine-dependent processing times.

Let $\mathcal{V}_t^{\mathrm{sch}} \subseteq \mathcal{V}_x$ denote the activities already scheduled in the reference state s_t. After selecting the prefix $\boldsymbol{a}_{t:t+h-1}$, the scheduled set becomes

$$\mathcal{V}_{t+h}^{\mathrm{sch}} = \mathcal{V}_t^{\mathrm{sch}} \cup \{a_{t+v} : 0 \le v < h\}. \tag{21}$$

The mask rules below specify the entries of $m_{t+h} = \mathcal{M}_x(s_t, \boldsymbol{a}_{t:t+h-1})$, as defined in Equation 2. They determine which activity may be scheduled next, rather than which activity may start at a current physical time. Here, $\mathbf{1}[\cdot]$ denotes the indicator function, which equals one when its condition holds and zero otherwise.

For a selected activity, $\mathcal{T}_x$ returns the successor state by recording its start and completion times, adding it to the scheduled set, and updating the relevant scheduling information. Previously assigned start times remain unchanged. The following subsections distinguish the constraints defining Γ_x from the placement rules adopted by this constructive interface.

A.1 JOB-SHOP SCHEDULING

Each job $j \in \mathcal{J}_x$ follows a prescribed route through the machines. For the classical instances considered here, each job contains M operations and visits each machine once. We index an operation by $i = (j, k)$, where k is its position within the job. The activity set is $\mathcal{V}_x = \{(j, k) : j \in \mathcal{J}_x,\ k = 1, \dots, M\}$, containing all operations, with $N_x = |\mathcal{J}_x|M$. The machine required by operation i is denoted by μ_i.

Feasibility. The indicator satisfies $\Gamma_x(S) = 1$ exactly when within-job precedence and machine non-overlap hold:

$$\begin{aligned} S_i &\ge 0, && \forall i \in \mathcal{V}_x, \\ C_{j,k} &\le S_{j,k+1}, && \forall j \in \mathcal{J}_x, \quad k = 1, \dots, M-1, \\ C_i \le S_{i'} \quad &\text{or} \quad C_{i'} \le S_i, && \forall \{i, i'\} \in \binom{\mathcal{V}_x}{2} \text{ with } \mu_i = \mu_{i'}. \end{aligned} \tag{22}$$

The final condition prevents two operations from occupying the same machine simultaneously.

State. For a fixed instance x, the exact state is represented as $s_t = \left(k_t, r_t, b_t, \sigma_t, S^{(t)}\right)$. Here, $k_t \in \{0, \dots, M\}^{|\mathcal{J}_x|}$ records the number of scheduled operations in each job. The vectors $r_t \in \mathbb{R}_{\ge 0}^{|\mathcal{J}_x|}$ and $b_t \in \mathbb{R}_{\ge 0}^{M}$ record job and machine readiness times, respectively. Specifically, $r_{t,j}$ is the completion time of the last scheduled operation of job j, and $b_{t,\mu}$ is the end of the existing schedule on machine μ. Both are initialized to zero. The indicator $\sigma_t \in \{0, 1\}^{N_x}$ identifies scheduled operations, while $S^{(t)} \in \mathbb{R}_{\ge 0}^{N_x}$ stores their assigned start times. Entries for unscheduled operations are set to zero and interpreted only through σ_t.

Action. The action $a_t = (j, k) \in \mathcal{A}_x(m_t)$ selects the next unscheduled operation of job j, with $k = [k_t]_j + 1$. Its required machine $\mu_{(j,k)}$ is fixed by x, while its start time is determined by $\mathcal{T}_x$.

Mask update. An operation is selectable if it remains unscheduled and its within-job predecessor has already been scheduled:

$$\begin{aligned} [m_{t+h}]_{j,k} = \mathbf{1}\Big[&(j, k) \notin \mathcal{V}_{t+h}^{\mathrm{sch}} \\ &\wedge \left(k = 1 \text{ or } (j, k-1) \in \mathcal{V}_{t+h}^{\mathrm{sch}}\right)\Big]. \end{aligned} \tag{23}$$

Thus, $\mathcal{M}_x$ exposes the next unscheduled operation of each unfinished job. The predecessor must have an assigned processing interval, but need not have physically completed at the point of selection.

Scheduling transition. We use deterministic append-only placement. At the corresponding exact state s_{t+h}, let $r_{t+h,j}$ denote the completion time of the last scheduled operation of job j, and let b_{t+h,μ_i} denote the end of the existing schedule on machine μ_i. Both are zero when no relevant operation has been scheduled. For a selected operation $i = (j, k)$, $\mathcal{T}_x$ assigns

$$S_i = \max\left(r_{t+h,j}, b_{t+h,\mu_i}\right). \tag{24}$$

It then updates the corresponding readiness times:

$$r_{t+h+1,j} = C_i, \qquad b_{t+h+1,\mu_i} = C_i. \tag{25}$$

Other readiness entries remain unchanged. Append-only placement specifies our constructive rule, not an additional requirement of Γ_x.

A.2 Permutation Flow-Shop Scheduling

In PFSP, all jobs follow the same machine route, with operation (j, k) processed on machine k. The activity set is $\mathcal{V}_x = \{(j, k) : j \in \mathcal{J}_x,\ k = 1, \ldots, M\}$, containing all operations, with $N_x = |\mathcal{J}_x|M$. Every machine must process the jobs according to one common permutation.

Feasibility. The indicator satisfies $\Gamma_x(S) = 1$ exactly when there exists a permutation $\boldsymbol{o} = (o_1, \ldots, o_{|\mathcal{J}_x|})$ of the jobs such that

$$\begin{aligned} S_{j,k} &\geq 0, && \forall j \in \mathcal{J}_x, \quad k = 1, \ldots, M, \\ C_{j,k} &\leq S_{j,k+1}, && \forall j \in \mathcal{J}_x, \quad k = 1, \ldots, M-1, \\ C_{o_r,k} &\leq S_{o_{r+1},k}, && r = 1, \ldots, |\mathcal{J}_x| - 1, \quad k = 1, \ldots, M. \end{aligned} \tag{26}$$

The final condition enforces the shared job order and consequently prevents overlap on each machine.

State. For a fixed instance x, the exact state is represented as $s_t = \left(k_t, r_t, b_t, \sigma_t, S^{(t)}, c_t, \boldsymbol{o}_t\right)$. Here, $k_t \in \{0, \ldots, M\}^{|\mathcal{J}_x|}$ records the number of scheduled operations in each job. The vectors $r_t \in \mathbb{R}_{\geq 0}^{|\mathcal{J}_x|}$ and $b_t \in \mathbb{R}_{\geq 0}^{M}$ record job and machine readiness times, respectively. Specifically, $r_{t,j}$ is the completion time of the last scheduled operation of job j, and $b_{t,k}$ is the end of the existing schedule on machine k. Both are initialized to zero. The indicator $\sigma_t \in \{0, 1\}^{N_x}$ identifies scheduled operations, while $S^{(t)} \in \mathbb{R}_{\geq 0}^{N_x}$ stores their assigned start times. Entries for unscheduled operations are set to zero and interpreted only through σ_t. The scalar $c_t \in \mathbb{R}_{\geq 0}$ records the current partial makespan, initialized to zero. Finally, $\boldsymbol{o}_t$ is an ordered sequence of distinct jobs from $\mathcal{J}_x$, recording the order in which their first-machine operations have been selected. It is initially empty.

Action. The action $a_t = (j, k) \in \mathcal{A}_x(m_t)$ selects the next unscheduled operation of job j, with $k = [k_t]_j + 1$. Its required machine is k, and its start time is determined by $\mathcal{T}_x$. Selecting $(j, 1)$ appends job j to $\boldsymbol{o}_t$. For $k > 1$, the mask $\mathcal{M}_x$ permits selection only when the within-job and shared-permutation predecessors have been scheduled. Each action therefore schedules one operation while preserving the common job order across machines.

Mask update. First-machine decisions establish the shared job permutation. Let $\boldsymbol{o}_{t+h}$ denote the partial permutation determined by the reference state and the selected prefix, with o_r its r-th job. Any job whose first operation remains unscheduled may extend this order:

$$[m_{t+h}]_{j,1} = \mathbf{1}\left[(j, 1) \notin \mathcal{V}_{t+h}^{\text{sch}}\right]. \tag{27}$$

For a job already in the partial permutation, a later operation is selectable only after its within-job predecessor and the corresponding operations of all earlier jobs have been scheduled:

$$\begin{aligned} [m_{t+h}]_{o_r,k} = \mathbf{1}\Big[&(o_r, k) \notin \mathcal{V}_{t+h}^{\text{sch}} \\ &\wedge \left(\{(o_r, k-1)\} \cup \{(o_v, k) : 1 \leq v < r\}\right) \subseteq \mathcal{V}_{t+h}^{\text{sch}}\Big], \end{aligned} \tag{28}$$

for $k = 2, \ldots, M$ and each valid position r in $\boldsymbol{o}_{t+h}$. Later operations of jobs not yet introduced into the permutation remain masked.

Scheduling transition. For a selected operation (j, k), $\mathcal{T}_x$ assigns

$$S_{j,k} = \max\left(r_{t+h,j}, b_{t+h,k}\right). \tag{29}$$

The two readiness times correspond to the preceding operation of the same job and the preceding job on the same machine under the shared permutation. As in JSSP, both are zero when the corresponding predecessor is absent. Selecting a first-machine operation appends its job to the partial permutation. Later operations preserve that order. The transition records the assigned interval and updates the corresponding readiness times to $C_{j,k}$. Together, the mask and placement rules construct a permutation schedule through operation-level decisions.

A.3 Flexible Job-Shop Scheduling

In FJSP, job $j \in \mathcal{J}_x$ contains K_j operations in a fixed precedence order. Operation $i = (j, k)$ can be processed by any machine in its eligibility set E_i. Its processing time on machine $\mu \in E_i$ is $p_{i,\mu}$. The activity set is $\mathcal{V}_x = \{(j, k) : j \in \mathcal{J}_x,\ k = 1, \ldots, K_j\}$, with $N_x = \sum_{j \in \mathcal{J}_x} K_j$.

Feasibility. An FJSP schedule specifies start times S_i and machine assignments $\alpha_i \in E_i$. Writing $C_i = S_i + p_{i,\alpha_i}$, it is feasible exactly when

$$\begin{aligned} S_i &\geq 0, && \forall i \in \mathcal{V}_x, \\ C_{j,k} &\leq S_{j,k+1}, && \forall j \in \mathcal{J}_x, \quad k = 1, \ldots, K_j - 1, \\ C_i \leq S_{i'} \quad \text{or} \quad C_{i'} &\leq S_i, && \forall i \neq i' \text{ with } \alpha_i = \alpha_{i'}. \end{aligned} \tag{30}$$

Thus, the selected assignments determine both processing times and which operations compete for the same machine.

State. The exact state extends the JSSP state as $s_t = (k_t, r_t, b_t, \sigma_t, S^{(t)}, c_t, \alpha_t)$. Here, $[k_t]_j \in \{0, \ldots, K_j\}$ counts the operations already scheduled for job j. The vectors r_t and b_t record job and machine readiness times. The mask σ_t identifies scheduled operations, $S^{(t)}$ stores their start times, and c_t is the current partial makespan. The additional field α_t stores the machine assigned to each scheduled operation. Both $S^{(t)}$ and α_t are interpreted only for operations marked by σ_t.

Action. An action $a_t = ((j, k), \mu)$ selects the next operation of job j, where $k = [k_t]_j + 1$, and assigns it to an eligible machine $\mu \in E_{j,k}$. Unlike JSSP and PFSP, the machine is part of the decision. The start time is determined by $\mathcal{T}_x$.

Mask update. The FJSP mask is indexed by operation–machine pairs. After the selected prefix, a pair is selectable if the operation remains unscheduled, its within-job predecessor has been scheduled, and the machine is eligible:

$$\begin{aligned} [m_{t+h}]_{(j,k),\mu} = \mathbf{1}\Big[(j, k) \notin \mathcal{V}^{\text{sch}}_{t+h} \ \wedge\ \mu \in E_{j,k} \\ \wedge \left(k = 1 \text{ or } (j, k-1) \in \mathcal{V}^{\text{sch}}_{t+h}\right)\Big]. \end{aligned} \tag{31}$$

Because only one operation per unfinished job satisfies the precedence condition, the implementation may equivalently index the mask by job–machine pairs (j, μ). Machine availability does not restrict selection, since an eligible operation can be placed later.

Scheduling transition. For a selected action $((j, k), \mu)$, $\mathcal{T}_x$ uses deterministic append-only placement:

$$S_{j,k} = \max\left(r_{t+h,j}, b_{t+h,\mu}\right), \qquad C_{j,k} = S_{j,k} + p_{j,k,\mu}. \tag{32}$$

It records $\alpha_{j,k} = \mu$, marks (j, k) as scheduled, and updates

$$r_{t+h+1,j} = C_{j,k}, \qquad b_{t+h+1,\mu} = C_{j,k}. \tag{33}$$

Other readiness entries remain unchanged. As in JSSP, append-only placement is the constructive rule used by $\mathcal{T}_x$, not an additional constraint in the definition of a feasible FJSP schedule.

B Optimality Preservation of the Constructive Scheduling Interfaces

The interfaces in Appendix A use append-only placement and need not generate every feasible timing assignment. We show that they nevertheless preserve the optimal makespan for JSSP, PFSP, and FJSP. Let $\mathcal{F}_x = \{S : \Gamma_x(S) = 1\}$ be the feasible scheduling space, and let $\mathcal{G}_x = \{S(\boldsymbol{a}) : \boldsymbol{a} \in \Pi_x\}$ be the set of terminal schedules generated by complete admissible action sequences. The claim is

$$\min_{\boldsymbol{a}\in\Pi_x} C_{\max}(S(\boldsymbol{a})) = \min_{S\in\mathcal{F}_x} C_{\max}(S). \tag{34}$$

Proof. Every complete admissible action sequence produces a feasible schedule under the masks and transitions in Appendix A. Thus, $\mathcal{G}_x \subseteq \mathcal{F}_x$, which gives

$$\min_{S\in\mathcal{F}_x} C_{\max}(S) \le \min_{\boldsymbol{a}\in\Pi_x} C_{\max}(S(\boldsymbol{a})). \tag{35}$$

It remains to establish the reverse inequality.

Fix an arbitrary $S \in \mathcal{F}_x$ and construct a directed graph $G(S) = (\mathcal{V}_x, E(S))$. For JSSP and FJSP, include the within-job precedence arcs and, on each machine, an arc between consecutive operations in their execution order in S. For FJSP, use the machine assignments α_i specified by S when forming these machine-order arcs. For PFSP, feasibility under Equation 26 provides a common job permutation $\boldsymbol{o} = (o_1, \ldots, o_{|\mathcal{J}_x|})$. Include the within-job arcs $(o_r, k) \to (o_r, k+1)$ and the machine-order arcs $(o_r, k) \to (o_{r+1}, k)$.

Every arc $q \to i$ represents a relation $C_q \le S_i$ satisfied by S. Since processing times are positive, $G(S)$ cannot contain a directed cycle. Let $(v_0, \ldots, v_{N_x-1})$ be a topological ordering of its operations. Use this order to define an action sequence $\boldsymbol{a}$: set $a_t = v_t$ for JSSP and PFSP, and $a_t = (v_t, \alpha_{v_t})$ for FJSP.

This sequence is admissible under the masks in Appendix A. For JSSP and FJSP, the within-job predecessor of each selected operation has already appeared. In FJSP, the retained assignment also satisfies $\alpha_{v_t} \in E_{v_t}$. Hence Equations 23 and 31 permit the corresponding actions. For PFSP, the first-machine arcs cause those operations to be selected in the order $o_1, \ldots, o_{|\mathcal{J}_x|}$, reconstructing $\boldsymbol{o}$. Each later operation has its within-job predecessor and all earlier jobs' operations on the same machine before it in the topological order. It is therefore permitted by Equations 27 and 28. Consequently, $\boldsymbol{a} \in \Pi_x$.

Let $\widetilde{S} = S(\boldsymbol{a})$ be the schedule produced by the constructive transitions. For an operation i, let $P(i)$ be its direct predecessors in $G(S)$, with $\max_{q\in\varnothing} C_q = 0$. For FJSP, write $p_i = p_{i,\alpha_i}$; the construction retains this assignment. We prove by induction over the topological order that $\widetilde{S}_i \le S_i$ and $\widetilde{C}_i \le C_i$ for every $i \in \mathcal{V}_x$.

Suppose the claim holds for all $q \in P(i)$. The append-only transition places i at the maximum of its job and machine readiness times. Because the action order follows $G(S)$, these times are exactly the completion times of its direct predecessors, or zero when a predecessor is absent. By Equations 24, 29, and 32,

$$\begin{aligned} \widetilde{S}_i &= \max_{q\in P(i)} \widetilde{C}_q \le \max_{q\in P(i)} C_q \le S_i, \\ \widetilde{C}_i &= \widetilde{S}_i + p_i \le S_i + p_i = C_i. \end{aligned} \tag{36}$$

The induction also covers operations with no predecessors by the stated convention. Hence $C_{\max}(\widetilde{S}) \le C_{\max}(S)$.

Since S was arbitrary, every feasible schedule is weakly dominated in makespan by a constructive schedule. In particular, applying the argument to an optimal feasible schedule gives

$$\min_{\boldsymbol{a}\in\Pi_x} C_{\max}(S(\boldsymbol{a})) \le \min_{S\in\mathcal{F}_x} C_{\max}(S). \tag{37}$$

Combining Equations 35 and 37 proves Equation 34. □

The argument also establishes the stronger componentwise property: for every feasible schedule S, there is an admissible $\boldsymbol{a} \in \Pi_x$ such that

$$C_i(S(\boldsymbol{a})) \le C_i(S), \qquad \forall i \in \mathcal{V}_x. \tag{38}$$

C OBJECTIVES OF MSLPM

This appendix expands the training objectives introduced in Section 4.4. We follow the notation of the main text. All losses are defined over activity representations $z_t \in \mathbb{R}^{N_x \times d}$. Exact masks are supplied by $\mathcal{M}_x$ and are not predicted by the learned Dynamics model.

C.1 OBJECTIVE OF STAGE 1: LEARNING TO SOLVE

Stage 1 jointly trains the Encoder E_ϕ and Actor π_θ. For a training tuple $(s_t, a_t^{\mathrm{tr}}, \mathcal{P}) \sim \mathcal{D}_1$, let

$$z_t = E_\phi(s_t), \qquad m_t = \mathcal{M}_x(s_t, ()),$$

where $a_t^{\mathrm{tr}} \in \mathcal{P}$ and $\mathcal{P} \subseteq \mathcal{A}_x(m_t)$. Equation 18 is instantiated as

$$\mathcal{L}_{\mathrm{solve}}(\phi, \theta) = \mathbb{E}_{(s_t, a_t^{\mathrm{tr}}, \mathcal{P}) \sim \mathcal{D}_1} \Big[\mathcal{L}_{\mathrm{sup}} \left(z_t, m_t, a_t^{\mathrm{tr}}, \mathcal{P} \right) \Big]. \tag{39}$$

The supervised loss combines recorded-action supervision, certified-set supervision, and a separation objective:

$$\mathcal{L}_{\mathrm{sup}} = \lambda_{\mathrm{tr}} \mathcal{L}_{\mathrm{tr}} + \lambda_{\mathrm{set}} \mathcal{L}_{\mathrm{set}} + \lambda_{\mathrm{mar}} \mathcal{L}_{\mathrm{mar}}. \tag{40}$$

Teacher-certified action set. The set $\mathcal{P}$ contains selectable actions that preserve the ordering induced by the teacher schedule. Within each ordering group exposed by the problem adapter, activities are ordered by their teacher start time, completion time, and fixed activity index. Each activity receives the immediately preceding activity in this order as an additional teacher predecessor. At state s_t, a selectable activity belongs to $\mathcal{P}$ if its teacher predecessor has already been selected or if it has no such predecessor.

This construction guarantees that $\mathcal{P}$ is nonempty and that the recorded action satisfies $a_t^{\mathrm{tr}} \in \mathcal{P}$. It does not assume that all actions in $\mathcal{P}$ lead to an identical completed schedule. We use

$$\mathcal{N} := \mathcal{A}_x(m_t) \setminus \mathcal{P} \tag{41}$$

to denote the remaining selectable actions.

In addition to the recorded trajectory, Stage 1 constructs augmented trajectories by sampling actions from $\mathcal{P}$. Such samples contribute certified-set and margin supervision, but their sampled actions are not treated as preferred teacher labels.

Likelihood supervision. During the first training phase, recorded-action supervision is given by

$$\mathcal{L}_{\mathrm{tr}}^{\mathrm{CE}} = -\log \pi_\theta \left(a_t^{\mathrm{tr}} \mid z_t, m_t \right). \tag{42}$$

This term is applied only to samples from the recorded teacher trajectory.

Certified-set supervision maximizes the total probability assigned to $\mathcal{P}$:

$$\mathcal{L}_{\mathrm{set}}^{\mathrm{CE}} = -\log \sum_{i \in \mathcal{P}} \pi_\theta(i \mid z_t, m_t). \tag{43}$$

Unlike Equation 42, this objective does not specify a preference among the actions in $\mathcal{P}$.

Let $\ell_{t,i}$ denote the Actor logit of activity i as defined in Equation 9. When $\mathcal{N}$ is nonempty, the separation loss compares the strongest certified and non-certified actions:

$$\mathcal{L}_{\mathrm{mar}}^{\mathrm{set}} = \tau_{\mathrm{s}} \log \left[1 + \exp \left(\frac{\delta_{\mathrm{s}} + \max_{j \in \mathcal{N}} \ell_{t,j} - \max_{i \in \mathcal{P}} \ell_{t,i}}{\tau_{\mathrm{s}}} \right) \right]. \tag{44}$$

Here $\delta_{\mathrm{s}} \geq 0$ is the requested separation margin and $\tau_{\mathrm{s}} > 0$ is a smoothing temperature. We set $\mathcal{L}_{\mathrm{mar}}^{\mathrm{set}} = 0$ when $\mathcal{N} = \varnothing$.

The likelihood phase therefore uses

$$\mathcal{L}_{\mathrm{sup}}^{\mathrm{CE}} = \lambda_{\mathrm{tr}} \mathcal{L}_{\mathrm{tr}}^{\mathrm{CE}} + \lambda_{\mathrm{set}} \mathcal{L}_{\mathrm{set}}^{\mathrm{CE}} + \lambda_{\mathrm{mar}} \mathcal{L}_{\mathrm{mar}}^{\mathrm{set}}. \tag{45}$$

Ranking refinement. The second training phase replaces likelihood supervision with pairwise logit comparisons. The recorded-action term ranks the recorded action above the other certified actions:

$$\mathcal{L}_{\mathrm{tr}}^{\mathrm{pair}} = \frac{\sum_{j \in \mathcal{P} \setminus \{a_t^{\mathrm{tr}}\}} \mathrm{softplus}\left(\ell_{t,j} - \ell_{t,a_t^{\mathrm{tr}}}\right)}{\max\{1, |\mathcal{P}| - 1\} \log 2}. \tag{46}$$

The certified-set term ranks every certified action above every non-certified action:

$$\mathcal{L}_{\mathrm{set}}^{\mathrm{pair}} = \frac{\sum_{i \in \mathcal{P}} \sum_{j \in \mathcal{N}} \mathrm{softplus}\left(\ell_{t,j} - \ell_{t,i}\right)}{\max\{1, |\mathcal{P}|\,|\mathcal{N}|\} \log 2}. \tag{47}$$

An empty comparison set gives zero loss. Averaging over the comparisons reduces the dependence of each term on the number of selectable activities. The factor $\log 2$ makes an equal-logit pair contribute unit loss.

During refinement, the margin term is applied only when the requested separation has not been achieved:

$$\mathcal{L}_{\mathrm{mar}}^{\mathrm{viol}} = \begin{cases} \mathcal{L}_{\mathrm{mar}}^{\mathrm{set}}, & \mathcal{N} \neq \varnothing, \\ & \max_{i \in \mathcal{P}} \ell_{t,i} < \max_{j \in \mathcal{N}} \ell_{t,j} + \delta_{\mathrm{s}}, \\ 0, & \text{otherwise.} \end{cases} \tag{48}$$

The refinement objective is

$$\mathcal{L}_{\mathrm{sup}}^{\mathrm{pair}} = \lambda_{\mathrm{tr}} \mathcal{L}_{\mathrm{tr}}^{\mathrm{pair}} + \lambda_{\mathrm{set}} \mathcal{L}_{\mathrm{set}}^{\mathrm{pair}} + \lambda_{\mathrm{mar}} \mathcal{L}_{\mathrm{mar}}^{\mathrm{viol}}. \tag{49}$$

Stage-1 settings. Stage 1 is trained for 24 epochs. The first 12 epochs use Equation 45 with

$$(\lambda_{\mathrm{tr}}, \lambda_{\mathrm{set}}, \lambda_{\mathrm{mar}}) = (1, 1, 0.5).$$

The remaining 12 epochs use Equation 49 with

$$(\lambda_{\mathrm{tr}}, \lambda_{\mathrm{set}}, \lambda_{\mathrm{mar}}) = (2, 0.5, 1).$$

We set $\tau_{\mathrm{s}} = 0.1$. The separation margin is $\delta_{\mathrm{s}} = 0.1$ during the likelihood phase and decreases linearly from 0.1 to 0.05 during refinement. The same objective schedule is used for JSSP and PFSP.

C.2 Objective of Stage 2: Learning to Imagine

Stage 2 freezes E_ϕ and π_θ and optimizes only F_ψ. The objective from Equation 20 is

$$\begin{aligned} \mathcal{L}_{\mathrm{imagine}}(\psi) := \mathbb{E}_{\mathcal{D}_2} \Bigg[& \\ & \sum_{h=1}^{K} w_{K,h} \Big(\lambda_{\mathrm{lat}} \mathcal{L}_{\mathrm{lat}}\left(\widehat{z}_{t+h}, z_{t+h}, m_{t+h}\right) + \mathcal{L}_{\mathrm{dec}}\left(\widehat{z}_{t+h}, z_{t+h}, m_{t+h}\right) \Big) \\ & + \lambda_{\mathrm{anc}} \mathcal{L}_{\mathrm{anc}}\left(s_t, a_t^{\mathrm{tr}}, m_t\right) \Bigg]. \end{aligned} \tag{50}$$

The first two losses supervise recursively imagined representations at every rollout depth. The final term separately supervises a one-step transition from a freshly encoded state.

Paired exact and imagined rollouts. Each rollout window begins from

$$z_t = E_\phi(s_t), \qquad \widehat{z}_t = z_t.$$

For $h = 0, \ldots, K-1$, the exact and imagined branches follow the same action a_{t+h}:

$$\begin{aligned} s_{t+h+1} &= \mathcal{T}_x(s_{t+h}, a_{t+h}), \\ z_{t+h+1} &= E_\phi(s_{t+h+1}), \\ \widehat{z}_{t+h+1} &= F_\psi\left(\widehat{z}_{t+h}, a_{t+h}, m_{t+h}\right), \\ m_{t+h+1} &= \mathcal{M}_x(s_{t+h+1}, ()). \end{aligned} \tag{51}$$

The exact branch supplies fixed successor representations and masks. The imagined branch is recursively conditioned on its own previous predictions.

Replay windows are generated by the frozen Stage-1 policy. At the beginning of training, each action is selected from either the recorded replay prefix or the frozen Actor evaluated on the imagined representation. Once an imagined action differs from the recorded prefix, the remainder of that window follows the imagined Actor. The exact branch applies the same selected actions, ensuring that z_{t+h} remains the exact target corresponding to the imagined prefix.

Latent alignment. Following self-predictive representation learning methods (Schwarzer et al., 2021; Ni et al., 2024), we align recursively predicted activity representations with exact successor encodings at every rollout depth. We use coordinate-wise smooth-L_1 regression. Define

$$\rho_{\mathrm{SL1}}(u) = \begin{cases} \frac{1}{2}u^2, & |u| < 1, \\ |u| - \frac{1}{2}, & |u| \geq 1. \end{cases} \tag{52}$$

For activity i at rollout depth h, the token-level error is

$$e_{h,i}\left(\widehat{z}_{t+h}, z_{t+h}\right) = \frac{1}{d}\sum_{k=1}^{d} \rho_{\mathrm{SL1}}\left([\widehat{z}_{t+h}]_{i,k} - [z_{t+h}]_{i,k}\right). \tag{53}$$

The exact successor mask gives greater weight to representations that will be consumed by the next Actor decision. For a minibatch $\mathcal{B} = \{1, \ldots, B\}$, define the selectable and background index sets

$$\begin{aligned} \mathcal{C}_h &:= \left\{ (b,i) \,\middle|\, i \in \mathcal{A}_{x_b}\left(m_{t+h}^{(b)}\right) \right\}, \\ \mathcal{R}_h &:= \left\{ (b,i) \,\middle|\, i \in \mathcal{V}_{x_b} \setminus \mathcal{A}_{x_b}\left(m_{t+h}^{(b)}\right) \right\}. \end{aligned} \tag{54}$$

When the corresponding set is nonempty, let

$$\begin{aligned} \mu_{\mathcal{C},h} &:= \frac{1}{|\mathcal{C}_h|}\sum_{(b,i)\in\mathcal{C}_h} e_{h,i}^{(b)}, \\ \mu_{\mathcal{R},h} &:= \frac{1}{|\mathcal{R}_h|}\sum_{(b,i)\in\mathcal{R}_h} e_{h,i}^{(b)}, \\ \mu_{\mathrm{all},h} &:= \frac{\displaystyle\sum_{b=1}^{B}\sum_{i\in\mathcal{V}_{x_b}} e_{h,i}^{(b)}}{\displaystyle\sum_{b=1}^{B} N_{x_b}}. \end{aligned} \tag{55}$$

The minibatch latent loss is

$$\mathcal{L}_{\mathrm{lat},h}^{\mathcal{B}} = \begin{cases} \dfrac{\gamma\mu_{\mathcal{C},h} + \mu_{\mathcal{R},h}}{\gamma + 1}, & \mathcal{C}_h \neq \varnothing \text{ and } \mathcal{R}_h \neq \varnothing, \\ \mu_{\mathrm{all},h}, & \text{otherwise.} \end{cases} \tag{56}$$

Here $\gamma > 0$ controls the relative contribution of selectable activity tokens. For a single example, this is the pointwise $\mathcal{L}_{\mathrm{lat}}(\widehat{z}_{t+h}, z_{t+h}, m_{t+h})$ used in the main text.

The groups in Equation 54 are determined by the exact mask m_{t+h}. They are distinct from the internal update positions $\mathcal{U}_{t+h-1}$ used to place Dynamics residuals.

Decision consistency. The frozen Actor evaluates the exact and imagined representations under the same exact mask. For each selectable activity, define

$$\begin{aligned} p_{h,i} &:= \pi_\theta\left(i \mid z_{t+h}, m_{t+h}\right), \\ \widehat{p}_{h,i} &:= \pi_\theta\left(i \mid \widehat{z}_{t+h}, m_{t+h}\right). \end{aligned} \tag{57}$$

Following the policy-distillation principle (Rusu et al., 2016; Voelcker et al., 2025), distribution matching uses the exact Actor distribution as the target:

$$\mathcal{L}_{\mathrm{KL},h} = \sum_{i \in \mathcal{A}_x(m_{t+h})} p_{h,i} \log \frac{p_{h,i}}{\widehat{p}_{h,i}}. \tag{58}$$

The target distribution $p_{h,i}$ is detached. Thus, the KL direction is from the exact policy to the imagined policy.

For a nonterminal successor, the exact Actor supplies the reference action

$$a^{\mathrm{ref}}_{t+h} = \operatorname*{arg\,max}_{i \in \mathcal{A}_x(m_{t+h})} \pi_\theta\left(i \mid z_{t+h}, m_{t+h}\right). \tag{59}$$

Ties follow the fixed ordering of $\mathcal{V}_x$. The reference action need not equal the action used to generate the rollout or the next recorded replay action.

The imagined Actor is trained to reproduce this decision using

$$\mathcal{L}_{\mathrm{act},h} = -\log \pi_\theta\left(a^{\mathrm{ref}}_{t+h} \mid \widehat{z}_{t+h}, m_{t+h}\right). \tag{60}$$

Let $\widehat{\ell}_{t+h,i}$ denote the Actor logit obtained from $\widehat{z}_{t+h}$ under m_{t+h}. When more than one activity is selectable, the top-1 separation term is

$$\mathcal{L}_{\mathrm{top},h} = \tau_{\mathrm{d}} \log\left[1 + \exp\left(\frac{\delta_{\mathrm{d}} + \max\limits_{j \in \mathcal{A}_x(m_{t+h}) \setminus \{a^{\mathrm{ref}}_{t+h}\}} \widehat{\ell}_{t+h,j} - \widehat{\ell}_{t+h,a^{\mathrm{ref}}_{t+h}}}{\tau_{\mathrm{d}}}\right)\right]. \tag{61}$$

Here $\delta_{\mathrm{d}} \geq 0$ is the decision margin and $\tau_{\mathrm{d}} > 0$ is its smoothing temperature. We set $\mathcal{L}_{\mathrm{top},h} = 0$ when only one activity is selectable.

The pointwise decision loss from the main text is

$$\mathcal{L}_{\mathrm{dec}}\left(\widehat{z}_{t+h}, z_{t+h}, m_{t+h}\right) = \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL},h} + \lambda_{\mathrm{act}}\mathcal{L}_{\mathrm{act},h} + \lambda_{\mathrm{top}}\mathcal{L}_{\mathrm{top},h}. \tag{62}$$

For terminal masks, all three terms are zero. In minibatch training, KL and cross-entropy are averaged over examples with at least one selectable activity. The top-1 term is averaged over examples with at least two selectable activities. An empty eligible group contributes zero.

One-step anchoring. The one-step anchor uses the recorded replay action a^{tr}_t and starts from the freshly encoded representation z_t. Its exact and imagined successors are

$$\begin{aligned} s^{\mathrm{anc}}_{t+1} &= \mathcal{T}_x\left(s_t, a^{\mathrm{tr}}_t\right), \\ z^{\mathrm{anc}}_{t+1} &= E_\phi\left(s^{\mathrm{anc}}_{t+1}\right), \\ \widehat{z}^{\mathrm{anc}}_{t+1} &= F_\psi\left(z_t, a^{\mathrm{tr}}_t, m_t\right), \\ m^{\mathrm{anc}}_{t+1} &= \mathcal{M}_x\left(s_t, (a^{\mathrm{tr}}_t)\right). \end{aligned} \tag{63}$$

The anchor successor may differ from the first successor of the recurrent branch because the recurrent branch may select an imagined Actor action.

The anchor applies the latent and decision losses above to

$$\left(\widehat{z}^{\mathrm{anc}}_{t+1}, z^{\mathrm{anc}}_{t+1}, m^{\mathrm{anc}}_{t+1}\right).$$

It additionally normalizes each token error by the magnitude of its exact one-step change. Define

$$\sigma^{\mathrm{anc}}_i = \max\left\{\kappa, \frac{\left\|[z^{\mathrm{anc}}_{t+1}]_i - [z_t]_i\right\|_2}{\sqrt{d}}\right\}, \tag{64}$$

where $\kappa > 0$ prevents amplification when the exact change is small. The scale is detached from gradient computation.

The normalized token-level change error is

$$e^{\mathrm{anc}}_{\Delta,i} = \frac{1}{d}\sum_{k=1}^{d} \rho_{\mathrm{SL1}}\left(\frac{[\widehat{z}^{\mathrm{anc}}_{t+1} - z_t]_{i,k} - [z^{\mathrm{anc}}_{t+1} - z_t]_{i,k}}{\sigma^{\mathrm{anc}}_i}\right). \tag{65}$$

Both branches start from the same z_t. Therefore, the numerator equals the normalized successor-prediction error, while the exact latent change determines its scale.

The minibatch loss $\mathcal{L}^{\mathcal{B}}_{\Delta z}$ applies the pooling rule from Equation 56, replacing $e_{h,i}$ with $e^{\mathrm{anc}}_{\Delta,i}$ and using m^{anc}_{t+1} to form the selectable and background groups.

The complete anchor is

$$\begin{aligned}\mathcal{L}_{\mathrm{anc}}\left(s_t, a^{\mathrm{tr}}_t, m_t\right) =& \lambda^{\mathrm{anc}}_{\mathrm{lat}}\mathcal{L}^{\mathrm{anc}}_{\mathrm{lat}} + \lambda^{\mathrm{anc}}_{\Delta z}\mathcal{L}_{\Delta z} \\ &+ \lambda^{\mathrm{anc}}_{\mathrm{KL}}\mathcal{L}^{\mathrm{anc}}_{\mathrm{KL}} + \lambda^{\mathrm{anc}}_{\mathrm{act}}\mathcal{L}^{\mathrm{anc}}_{\mathrm{act}} + \lambda^{\mathrm{anc}}_{\mathrm{top}}\mathcal{L}^{\mathrm{anc}}_{\mathrm{top}}.\end{aligned} \tag{66}$$

The superscript anc indicates that the corresponding loss is evaluated using the anchor successor representations and mask from Equation 63. In particular, the anchor reference action is obtained by evaluating the exact Actor on z^{anc}_{t+1}.

Rollout-depth weighting. The recurrent objective averages supervision over the full rollout and gives additional weight to the first four predictions when $K \geq 5$:

$$w_{K,h} = \frac{1}{K} + \frac{2.5}{4}\mathbf{1}[K \geq 5]\mathbf{1}[h \leq 4], \qquad h = 1, \ldots, K. \tag{67}$$

The additional prefix weight is not renormalized. No geometric depth decay is used.

Stage-2 curriculum and coefficients. Stage 2 is trained for 48 epochs. Epochs 1–4, 5–8, 9–12 and 13–48 use training horizons

$$K = 1,\ 2,\ 3,\ 5,$$

respectively. The probability of selecting the frozen Actor's imagined action is 0.25, 0.25, 0.50, and 0.75 during the first four epochs and is 1 thereafter. Once a rollout deviates from its recorded prefix, all remaining actions in that window are selected by the imagined Actor.

Table 5 reports the effective coefficients after fixed implementation averaging. The anchor coefficients are applied inside $\mathcal{L}_{\mathrm{anc}}$, whose outer weight is $\lambda_{\mathrm{anc}} = 1$.

Both scheduling problems use

$$\gamma = 64, \qquad \delta_{\mathrm{d}} = 0.2, \qquad \tau_{\mathrm{d}} = 0.1, \qquad \kappa = 0.1.$$

Gradient flow. The exact successor encodings, exact Actor distributions, reference actions, and normalization scales are detached. Discrete action selection and the exact scheduling functions receive no gradients. The parameters of E_ϕ and π_θ remain frozen throughout Stage 2.

The Actor remains differentiable with respect to its imagined latent input. Consequently, the decision loss propagates through the frozen Actor to $\widehat{z}_{t+h}$ and then through the complete imagined rollout to F_ψ. Intermediate imagined representations are not detached. Both latent alignment and decision consistency therefore optimize only the Dynamics parameters ψ.

D Experimental Settings Detail

D.1 Benchmarks and Data Splits

Training and validation instances are generated independently for each problem and are disjoint from all benchmark test instances. The benchmark test sets are used only for final evaluation and do not participate in training or checkpoint selection.

JSSP. The training set contains 2,000 generated instances distributed evenly across eight size groups: 15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20, and 100×20. The validation set contains 80 independently generated instances, with ten instances per size. Testing uses the 80 Taillard instances from these size groups, Demirkol DMU01–DMU80, and Lawrence LA01–LA40 (Taillard, 1993; Demirkol et al., 1998; Lawrence, 1984).

PFSP. The training set contains 2,000 generated instances distributed evenly across eight size groups: 20×5, 20×10, 50×5, 50×10, 50×20, 100×5, 100×10, and 100×20. The validation set contains 80 independently generated instances, with ten instances per size. Testing uses all 80 Taillard instances from the selected size groups (Taillard, 1993).

Table 5: **Effective Stage-2 loss coefficients.**

Loss	Recurrent rollout	One-step anchor
Latent alignment	0.01	$0.10/3$
Normalized-change regression	0	$0.10/3$
Exact-to-imagined KL	1	2
Reference-action cross-entropy	12	8
Top-1 separation	4	4

FJSP. FJSP uses 1,000 generated training instances and a disjoint validation set of 20 generated instances. Testing is conducted on all ten Brandimarte instances, MK01–MK10, and the Hurink `edata`, `rdata`, and `vdata` benchmark sets (Brandimarte, 1993; Hurink et al., 1994). The training and validation pools contain no benchmark test instances.

D.2 Teacher Trajectory and Replay Construction

We construct two replay datasets with complementary roles. $\mathcal{D}_1$ provides state-level supervision for the Encoder and Actor in Stage 1, while $\mathcal{D}_2$ provides multi-step rollout windows for training the latent Dynamics model in Stage 2. The construction is shared across JSSP, PFSP, and FJSP, with FJSP actions additionally specifying a machine assignment.

Teacher-induced partial order. For each training or validation instance x, we obtain a feasible teacher schedule τ using OR-Tools CP-SAT (Perron et al., 2023). The schedule is then replayed in the exact simulator to obtain a legal dispatch sequence. Within each machine, scheduled operations are ordered lexicographically by start time, completion time, and fixed operation index. The immediately preceding operation in this order defines the teacher predecessor pred_τ.

Let $\mathcal{S}_t$ denote the operations selected before state s_t, and let $\mathcal{A}_x(m_t)$ denote the feasible action set under the exact simulator mask m_t. For JSSP and PFSP, the certified positive set is

$$\mathcal{P}_t = \{a \in \mathcal{A}_x(m_t) \mid \mathrm{pred}_\tau(a) = \varnothing \text{ or } \mathrm{pred}_\tau(a) \in \mathcal{S}_t\},$$

and the negative set is

$$\mathcal{N}_t = \mathcal{A}_x(m_t) \setminus \mathcal{P}_t.$$

Hence, $\mathcal{P}_t$ contains every feasible action that preserves the machine-order partial order induced by τ. This set-valued target avoids treating an arbitrary linearization of independent operations as the unique correct sequence.

For FJSP, an action is an operation–machine pair $a = (o, \mu)$. Let $E_x(o)$ denote the eligible machines of operation o, and let $\mu_\tau(o)$ denote its teacher-assigned machine. The feasible action set is

$$\mathcal{A}_x^{\mathrm{FJSP}}(m_t) = \{(o_t(j), \mu) \mid j \text{ is unfinished},\ \mu \in E_x(o_t(j)),\ (o_t(j), \mu) \text{ is legal under } m_t\},$$

where $o_t(j)$ is the next operation of job j. The corresponding positive set is

$$\mathcal{P}_t^{\mathrm{FJSP}} = \left\{(o, \mu) \in \mathcal{A}_x^{\mathrm{FJSP}}(m_t) \mid \mu = \mu_\tau(o),\ \mathrm{pred}_\tau(o) = \varnothing \text{ or } \mathrm{pred}_\tau(o) \in \mathcal{S}_t\right\}.$$

Thus, an FJSP action is positive only if it preserves both the teacher machine assignment and the induced machine order. Its negative set is $\mathcal{N}_t^{\mathrm{FJSP}} = \mathcal{A}_x^{\mathrm{FJSP}}(m_t) \setminus \mathcal{P}_t^{\mathrm{FJSP}}$. For batched computation, a job–machine action is flattened as

$$\mathrm{flat}(j, \mu) = jM_{\mathrm{pad}} + \mu,$$

where M_{pad} is the padded number of machines.

Stage 1 replay dataset. The dataset $\mathcal{D}_1$ is constructed by replaying each teacher sequence in the exact simulator and recording

$$\mathcal{D}_1 = \left\{\left(x, s_t, m_t, a_t^{\mathrm{tr}}, \mathcal{P}_t\right)\right\},$$

where $a_t^{\mathrm{tr}} \in \mathcal{P}_t$ is the recorded teacher action. Because each state is recorded after computing the exact feasibility mask, every stored action is executable by construction.

To expose the model to alternative valid linearizations, we additionally generate DAG-consistent trajectories by sampling the next action from $\mathcal{P}_t$ and applying it in the exact simulator. The sampled action is used only to reach a new certified state and is not interpreted as the unique preferred label. Consequently, augmented states retain positive-set supervision but do not receive unique-action supervision.

For FJSP, each replay row additionally stores the legal job–machine mask, the operation–machine eligibility mask, and the next-operation index of every job. In the current configuration, each instance contributes one original teacher replay and one DAG-consistent augmented replay. We retain at most 40 uniformly spaced states from each replay.

Stage 2 rollout dataset. After Stage 1, the Encoder E_ϕ and Actor π_θ are frozen. For JSSP and PFSP, we generate complete greedy trajectories by applying the frozen Actor to exact simulator states while masking infeasible actions. For FJSP, we use the feasible CP-SAT trajectory with the smallest makespan for each instance. In both cases, the selected action sequence is replayed in the exact simulator to construct windows of length K:

$$\mathcal{D}_2 = \{(s_{t:t+K}, m_{t:t+K}, a_{t:t+K-1}, \mathcal{P}_{t:t+K})\}.$$

Each window contains $K+1$ exact states, K actions, the associated feasibility masks, and the certified positive sets reconstructed from the stored schedule. Candidate starting positions are distributed across the complete trajectory. Training and validation instances are separated before window extraction, ensuring that windows from the same instance never appear in both splits.

For FJSP, the window representation additionally includes the operation–machine eligibility masks and next-operation indices:

$$\mathcal{W}_t^{\mathrm{FJSP}} = \left(s_{t:t+K}, m^{\mathrm{act}}_{t:t+K}, e_{t:t+K}, q_{t:t+K}, a_{t:t+K-1}, \mathcal{P}^{\mathrm{FJSP}}_{t:t+K}\right).$$

To reduce storage, exact FJSP states are retained at the initial state, every 16 simulator steps, and the terminal state. Intermediate states are reconstructed by restoring the nearest preceding anchor and replaying the stored actions. We use $K = 10$ and sample only windows for which at least one legal action remains after the supervised horizon, thereby avoiding an all-masked terminal Actor input.

Exact and imagined roll-ins. During Stage 2 training, the exact and imagined branches start from the same root encoding. Given a shared selected action $\tilde{a}_{t+h}$, the exact branch applies the simulator transition and re-encodes the successor:

$$s^{\mathrm{ex}}_{t+h+1} = T_x\left(s^{\mathrm{ex}}_{t+h}, \tilde{a}_{t+h}\right), \qquad z^{\mathrm{ex}}_{t+h+1} = E_\phi\left(s^{\mathrm{ex}}_{t+h+1}\right).$$

The imagined branch instead applies the learned latent transition recursively:

$$\widehat{z}_{t+h+1} = F_\psi\left(\widehat{z}_{t+h}, \widetilde{a}_{t+h}, m^{\mathrm{act}}_{t+h}\right).$$

The exact encoding is used only as a training target and is not provided as an input to the Actor or Dynamics model after the root state.

At each step, $\tilde{a}_{t+h}$ is selected from either the recorded replay or the greedy frozen Actor according to the on-policy curriculum. Actor actions are computed from the imagined latent state and filtered by the exact feasibility mask. Once an Actor action deviates from the recorded sequence, both branches follow the resulting on-policy action sequence for all remaining steps. This keeps the exact and imagined branches aligned even after they leave the original replay trajectory.

The Dynamics model predicts latent representations but does not learn action legality. Feasibility masks, eligibility masks, next-operation indices, and other symbolic fields are updated using the exact transition contract. Accordingly, the imagined symbolic mask is required to match the exact successor mask:

$$\widehat{m}^{\mathrm{act}}_{t+h+1} = m^{\mathrm{act}}_{t+h+1}.$$

After an on-policy deviation, the exact feasibility mask remains authoritative, while the positive set derived from the original teacher trajectory is no longer interpreted as a unique action label.

D.3 BASELINE SETTINGS

We compare MSLPM with three categories of baselines: rule-based greedy methods, learning-based neural constructive methods, and solver-based methods. For each problem class, the baselines are selected to cover conventional dispatching heuristics, representative neural constructive policies, and a general-purpose constraint-programming solver. Rule-based methods are independently reproduced in our experimental environment. Learning-based baselines are evaluated using the officially released implementations and pretrained checkpoints whenever applicable; when the released models do not directly cover a required evaluation setting, we use the corresponding released implementation for compatibility retraining or local reimplementation, as specified below. No learning-based baseline is fine-tuned on the test instances. OR-Tools CP-SAT (Perron et al., 2023) is used as the solver-based baseline.

Unless otherwise specified, neural constructive policies are evaluated by deterministic greedy decoding. The same decoding protocol is used for MSLPM and the neural baselines so that the comparison does not introduce an additional sampling budget for any method.

JSSP. Rule-based baselines. We use LPT (Longest Processing Time) and MOR (Most Operations Remaining) as classical dispatching-rule baselines. Both rules are independently reproduced locally. At each construction step, the rule assigns priority to the currently feasible operations and greedily selects the highest-priority operation. LPT prioritizes an eligible operation according to its processing time, whereas MOR prioritizes the job with the largest number of unscheduled operations remaining. These rules do not perform look-ahead search, learned inference, or test-instance-specific parameter fitting.

Learning-based baselines. We compare L2D, SLIM, and ReSCHED as learning-based neural constructive methods (Zhang et al., 2020; Corsini et al., 2024; Xiao et al., 2026). All three methods are evaluated using their officially released pretrained checkpoints. We do not retrain or fine-tune these models on the JSSP test instances.

The released checkpoints do not necessarily provide an exact-size model for every benchmark setting considered in our experiments. To avoid introducing target-size training into the comparison, we adopt the following direct size-generalization protocol: if an exact-size checkpoint is available, it is used directly; otherwise, we use the largest available checkpoint whose training size does not exceed the target size. The selected checkpoint is applied directly to the target instances without additional optimization or test-set adaptation. Therefore, a result obtained on a larger target size represents direct size generalization from the released model rather than performance after target-size retraining.

Solver baseline. We use OR-Tools CP-SAT as the solver-based baseline. CP-SAT is given a maximum wall-clock time limit of 5 s for each instance. The best feasible schedule obtained before termination is recorded as the CP-SAT solution for that instance. No additional solver-specific post-processing or instance-dependent time allocation is used.

Benchmarks and reference solutions. The JSSP evaluation covers the LA, TA, and DMU benchmark families, containing 40, 80, and 80 instances, respectively. All methods are evaluated on the same benchmark instances.

For each test instance, the makespan produced by a method is compared with the corresponding benchmark best-known solution (BKS). The JSSP optimality gap is therefore defined relative to the benchmark BKS for all methods, including CP-SAT. The 5 s limit controls the computational budget of the CP-SAT baseline but does not replace the BKS with the CP-SAT solution as the evaluation reference.

PFSP. Rule-based baselines. We use SPT (Shortest Processing Time) and LPT (Longest Processing Time) as rule-based greedy baselines. Both rules are independently reproduced locally. At each construction step, the rules greedily prioritize jobs according to their processing-time criterion and generate a complete permutation schedule without learned parameters or search-based refinement.

Learning-based baselines. We compare Graph-IL and PFSPNet as learning-based neural constructive baselines. Their implementations are reproduced locally because the released resources do not

directly provide a complete set of pretrained models covering all machine-count settings required by our evaluation.

To separate machine-count generalization from job-count generalization, we maintain one model for each machine-count setting: 5 machines, 10 machines, and 20 machines. The checkpoint associated with a given machine count is applied to all evaluated job counts with that machine count. Specifically, the 5-machine model is used for 20×5, 50×5, and 100×5; the 10-machine model is used for 20×10, 50×10, and 100×10; and the 20-machine model is used for 50×20 and 100×20.

For Graph-IL, the released 5-machine checkpoint is used directly. Because compatible released checkpoints are not available for the 10- and 20-machine settings, the corresponding models are obtained through compatibility retraining using the released implementation. This retraining is performed to obtain a model compatible with the required machine count and is not performed on the Taillard test instances.

For PFSPNet, no directly applicable set of official pretrained weights is available for the required settings. We therefore obtain the 5-, 10-, and 20-machine models through our local actor–critic reimplementation of PFSPNet. The models are trained on randomly generated PFSP instances and are subsequently evaluated on the Taillard benchmark. The Taillard test instances are not used for model training, checkpoint selection, or test-set fine-tuning.

Solver baseline. OR-Tools CP-SAT is used as the solver baseline with a maximum wall-clock time limit of 5 s per instance. The best feasible permutation schedule found within this budget is recorded as the solver result.

Benchmarks and reference solutions. We evaluate the following eight Taillard PFSP configurations: 20×5, 20×10, 50×5, 50×10, 50×20, 100×5, 100×10, and 100×20. Each configuration contains ten test instances.

All learning-based methods and MSLPM are evaluated on exactly the same Taillard instances under the same greedy decoding protocol. For every instance, the reported PFSP makespan is compared with the official Taillard BKS. The PFSP optimality gap is therefore measured against the Taillard BKS for every method, including CP-SAT. The 5 s CP-SAT limit only constrains the solver baseline and does not define the gap reference.

FJSP. Rule-based baselines. We use three composite priority dispatching rules: FIFO-SPT, MOR-EET, and MWKR-SPT. These rules combine a job/operation sequencing priority with a machine-selection priority and are independently reproduced locally.

FIFO-SPT uses FIFO (First-In-First-Out) as the job/operation priority and SPT (Shortest Processing Time) for the corresponding machine-selection decision. FIFO prioritizes the operation that has been waiting or become available earliest, while SPT selects the eligible operation–machine assignment with the shortest processing duration.

MOR-EET uses MOR (Most Operations Remaining) as the primary job/operation priority and EET (Earliest End Time) for machine selection. MOR prioritizes the job with the largest number of remaining operations. Among the feasible machine assignments associated with the selected priority, EET selects the machine that can complete the operation earliest.

MWKR-SPT uses MWKR (Most Work Remaining) as the primary job/operation priority and SPT for machine selection. MWKR prioritizes the job with the largest remaining workload, measured from the processing requirements of its remaining operations, while SPT selects the feasible machine assignment with the shortest processing duration.

These composite rules are applied greedily at every scheduling decision. They do not perform learned inference, population-based search, or post-hoc local improvement. Consequently, each rule produces one complete schedule through a deterministic sequence of dispatching decisions under the fixed implementation.

Learning-based baselines. We compare DANIEL-G, DOAGNN-G, and ReSCHED as learning-based neural constructive baselines (Wang et al., 2024; Zhao et al., 2025; Xiao et al., 2026). All three methods are evaluated using their officially released pretrained checkpoints. No model is fine-tuned on the Brandimarte or Hurink test instances.

Because the released checkpoints correspond to particular training configurations, we preserve the original pretrained models and explicitly specify the checkpoint used for each benchmark group rather than retraining a separate model for every test size. The checkpoint assignments in our experiments are:

- **DANIEL-G:** $20x10.pth$ is used for the Brandimarte instances, and `10x5.pth` is used for the Hurink edata, rdata, and vdata subsets.
- **DOAGNN-G:** `model_10_5_195.pt` is used for the Brandimarte benchmark and all three Hurink subsets.
- **ReSCHED:** `Old-SD1-10x05.pth` is used for the Brandimarte benchmark and all three Hurink subsets.

Thus, the reported neural-baseline results measure direct application and size generalization of the released policies under the stated checkpoint assignments. No test-instance solutions are used to adapt the model parameters.

Benchmark coverage. For the Brandimarte benchmark, we evaluate MK01–MK10. MK11–MK15 are not included in our evaluation. For the Hurink benchmark, we evaluate all three standard subsets: edata, rdata, and vdata. The same benchmark instances are used for the rule-based methods, neural baselines, MSLPM, and CP-SAT.

Solver baseline. OR-Tools CP-SAT is used with a maximum wall-clock time limit of 1,800 s per instance. For every FJSP test instance, we record the best feasible makespan obtained by CP-SAT within this 1800 s budget. Unlike the JSSP and PFSP experiments, this CP-SAT result is also used to construct the evaluation reference for FJSP.

More precisely, let C_i^{CP} denote the best feasible makespan returned by CP-SAT for instance i within 1800 s. We use C_i^{CP} as the instance-specific reference makespan and compute the reported FJSP gap of a method producing makespan C_i relative to this reference. Thus, the reference is determined independently for each test instance rather than by a single global reference value.

This solver-derived reference protocol is used only for FJSP. For JSSP, the benchmark BKS remains the reference, and for PFSP, the official Taillard BKS remains the reference. The different reference protocols are adopted because the FJSP experiments use the bounded CP-SAT solution as the common per-instance reference, whereas benchmark BKS values are used directly for JSSP and PFSP.

D.4 Training Settings

Training organization. JSSP, PFSP, and FJSP are trained independently. Although they share the same model backbone and training framework, they do not share model parameters, optimizer states, or checkpoints. Each problem follows a two-stage procedure. Stage 1 jointly trains the Encoder and Actor from random initialization using OR-Tools teacher trajectories and DAG or machine-order supervision. Stage 2 initializes from the corresponding Stage 1 checkpoint, freezes the Encoder and Actor, and trains only an identity-initialized Dynamics model through teacher-trajectory replay and progressively longer on-policy latent rollouts. Inference uses greedy open-loop decoding.

Stage 1. All three problems are trained for 24 epochs using three-GPU distributed data parallelism, with a batch size of 64 per GPU. We use AdamW (Loshchilov & Hutter, 2019) with weight decay 10^{-5}, gradient clipping at 1.0, and no dropout. The learning rate is 3×10^{-4} for epochs 1–12 and 5×10^{-5} for epochs 13–24. Stage 1 uses FP32 precision. The shared backbone has hidden dimension 256, four relational blocks, eight attention heads, and a two-layer candidate Actor. JSSP and PFSP use `global_total` feature normalization, whereas FJSP uses `route_local` normalization together with a task-specific operation–machine action adapter.

The Stage 1 objective combines trajectory-level cross-entropy and DAG set cross-entropy, each with weight 1.0. The top-1 margin loss has weight 0.5 during the first half of training and 1.0 during the second half. For JSSP and PFSP, the second phase additionally uses balanced pairwise training, size-aware PCGrad (Yu et al., 2020), and exact replay, while disabling DAG augmentation. FJSP does not use the size-aware PCGrad component. PFSP further uses size-bucketed sampling.

Stage 2. Dynamics is trained for 48 epochs for all three problems. Training uses three GPUs and a batch size of 32 per GPU, with size-specific batch adjustments for JSSP and PFSP. AdamW updates only Dynamics. The Actor learning rate is fixed to zero. The Dynamics learning rate is 10^{-4} for epochs 1–4, 5×10^{-5} for epochs 5–12, 3×10^{-5} for epochs 13–26, 2×10^{-5} for epochs 27–32, 10^{-5} for epochs 33–38, and 5×10^{-6} thereafter. Weight decay and gradient clipping remain 10^{-5} and 1.0, respectively. JSSP and PFSP use FP16 automatic mixed precision with gradient scaling, while FJSP uses FP32.

The Dynamics model has hidden dimension 512, three relational layers, residual scale 0.5, and identity initialization. JSSP additionally uses a symbolic successor mask without structured state channels. For JSSP and PFSP, the rollout curriculum uses horizons $1, 2, 3$ and 5. The on-policy probabilities during the first four epochs are 0.25, 0.25, 0.50, and 0.75, respectively, and are set to 1.0 thereafter.

For JSSP and PFSP, the Stage 2 loss weights are 64.0 for candidate-token prediction, 1.0 for policy consistency, 12.0 for top-1 policy prediction, 4.0 for the policy margin, 0.03 for latent transition prediction, and 1.0 for the teacher-forced anchor. Actor-root, set-behavior-cloning, and open-loop Actor losses are disabled. FJSP follows the same three Stage 2 supervision components—latent alignment, decision consistency, and one-step anchoring—with weights 64.0, 1.0, and 12.0 for candidate-token prediction, policy consistency, and top-1 policy prediction, respectively, while the remaining auxiliary terms are disabled.

D.5 Runtime and Hardware

All methods are evaluated on the same workstation equipped with an Intel Xeon E5-2686 v4 CPU (36 logical processors), an NVIDIA Tesla V100 GPU, and 32GB of RAM. Neural methods use a single GPU with a batch size of one, whereas all rule-based methods and CP-SAT run on the CPU. CP-SAT uses eight workers.

After warm-up, we measure the wall-clock time required to solve the complete benchmark and report the average time per instance. Timing includes state construction, neural inference, feasible-action masking, and schedule updates, but excludes model loading and dataset I/O. We use PyTorch 2.6.0 with CUDA 12.4 and OR-Tools 9.15.6755.

E Detailed Experimental Results

This appendix reports the disaggregated results underlying Tables 2 and 3. Each entry is reported as Gap/Time, where Gap is the mean optimality gap (%) and Time is the mean runtime in seconds. Results are averaged within each size group, and dataset-level averages are weighted by the corresponding number of test instances. All methods follow the same evaluation protocol and reference makespans as described in Section 5.

E.1 Detailed JSSP Results

Tables 6–8 report results for each size group in the Lawrence, Taillard, and Demirkol benchmarks. MSLPM achieves lower gaps than the compared greedy learning-based methods on most size groups. Increasing the planning horizon generally reduces runtime, while its effect on solution quality varies across instance sizes.

E.2 Detailed FJSP Results

Tables 9–12 provide detailed results on the Brandimarte and Hurink benchmark sets. Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. CP-SAT$_{1{,}800\,\mathrm{s}}$ defines the instance-wise reference makespan and is reported as Gap (mean reference makespan).

Table 6: **Detailed results on the Lawrence JSSP benchmarks.** Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. Each size group contains five instances.

Size	Rule-based		Greedy learning-based			Solver-based	MSLPM (ours)		
	LPT	MOR	L2D	SLIM	ReSCHED	CP-SAT$_{5s}$	$H=1$	$H=3$	$H=5$
10×5	33.65/0.001	14.60/0.001	29.77/0.056	9.99/0.035	15.91/0.521	0.000/0.101	10.39/0.429	10.35/0.260	12.24/0.206
10×10	33.32/0.001	11.95/0.002	36.39/0.113	9.63/0.068	12.20/1.051	0.000/1.379	9.45/0.844	9.42/0.482	12.36/0.410
15×5	22.96/0.002	5.00/0.002	13.56/0.108	3.80/0.061	5.49/0.749	0.000/0.054	1.94/0.614	2.28/0.355	2.43/0.321
15×10	38.39/0.003	23.38/0.003	41.36/0.241	13.33/0.121	17.83/1.557	0.862/4.269	8.63/1.269	8.95/0.711	10.54/0.616
15×15	43.71/0.008	18.65/0.008	32.77/0.363	12.98/0.216	18.21/2.387	1.516/4.829	11.51/0.975	10.87/0.543	11.51/0.471
20×5	18.18/0.003	6.71/0.005	14.00/0.172	3.55/0.091	5.17/1.061	0.000/0.135	1.61/0.822	0.97/0.483	0.66/0.428
20×10	45.35/0.004	21.87/0.005	33.49/0.360	15.59/0.173	17.91/2.398	1.502/3.663	10.27/1.730	8.25/1.031	9.90/0.833
30×10	29.78/0.011	7.72/0.016	22.68/0.607	3.07/0.309	9.21/3.554	0.000/2.225	2.14/2.622	1.33/1.552	2.63/1.246
Average	33.17/0.004	13.74/0.005	28.00/0.253	8.99/0.134	12.74/1.660	0.485/2.082	6.99/1.163	6.55/0.677	7.78/0.566

Table 7: **Detailed results on the Taillard JSSP benchmarks.** Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. Each size group contains ten instances.

Size	Rule-based		Greedy learning-based			Solver-based	MSLPM (ours)		
	LPT	MOR	L2D	SLIM	ReSCHED	CP-SAT$_{5s}$	$H=1$	$H=3$	$H=5$
15×15	41.22/0.004	20.53/0.008	36.90/0.459	15.05/0.203	19.06/2.415	2.487/5.021	14.00/1.466	12.97/0.849	14.06/0.699
20×15	46.18/0.009	23.56/0.014	42.31/0.565	16.16/0.257	23.25/3.113	8.289/5.039	13.63/1.275	13.46/0.727	13.84/0.654
20×20	40.86/0.012	21.71/0.022	42.73/0.895	14.70/0.340	21.80/4.202	8.933/5.074	13.36/1.760	13.95/0.982	14.70/0.840
30×15	49.13/0.018	22.83/0.030	40.28/1.052	17.35/0.503	24.35/5.390	13.560/5.021	14.74/2.035	13.49/1.192	14.20/0.956
30×20	50.27/0.025	24.97/0.045	43.17/1.884	20.85/0.633	24.84/6.806	17.405/5.025	16.69/2.683	17.02/1.481	16.01/1.277
50×15	41.21/0.048	17.37/0.081	28.24/2.795	11.76/0.977	16.08/8.442	12.694/5.094	11.91/3.362	10.53/1.924	11.09/1.706
50×20	43.74/0.069	17.68/0.114	31.99/5.237	14.80/1.287	17.58/10.292	15.297/5.049	10.76/4.921	10.40/2.786	10.85/2.412
100×20	31.92/0.240	9.15/0.417	17.15/25.906	7.26/3.566	8.50/23.043	7.956/5.161	6.43/14.234	6.13/7.130	6.44/5.458
Average	43.07/0.053	19.73/0.091	35.35/4.849	14.74/0.971	19.43/7.963	10.828/5.061	12.69/3.967	12.24/2.134	12.65/1.750

Table 8: **Detailed results on the Demirkol JSSP benchmarks.** Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. Each size group contains ten instances.

Size	Rule-based		Greedy learning-based			Solver-based	MSLPM (ours)		
	LPT	MOR	L2D	SLIM	ReSCHED	CP-SAT$_{5s}$	$H=1$	$H=3$	$H=5$
20×15	49.90/0.010	30.85/0.011	42.63/0.614	18.64/0.306	28.59/3.083	12.127/5.033	14.33/1.342	16.18/0.791	15.25/0.632
20×20	40.87/0.014	28.74/0.012	41.01/0.849	19.64/0.361	26.78/4.301	13.173/5.053	14.43/1.705	14.05/0.975	15.93/0.873
30×15	53.93/0.015	36.16/0.019	47.83/1.051	24.70/0.475	31.73/5.442	21.881/5.049	15.60/1.934	16.00/1.100	16.52/0.991
30×20	55.66/0.021	32.61/0.027	47.35/1.807	26.15/0.661	31.42/6.795	23.292/5.086	17.85/2.684	16.48/1.482	16.46/1.282
40×15	51.58/0.031	33.58/0.033	44.00/1.979	18.06/0.696	26.83/6.685	21.237/5.028	11.00/2.759	12.31/1.610	12.95/1.301
40×20	56.08/0.046	36.02/0.047	49.16/3.398	23.61/0.965	32.63/9.138	26.284/5.153	15.10/3.667	15.99/2.081	15.79/1.790
50×15	47.73/0.043	35.08/0.062	41.64/2.775	18.21/0.934	26.88/8.829	21.806/5.159	10.68/3.370	11.80/1.923	10.99/1.705
50×20	53.14/0.082	37.23/0.087	45.97/4.952	24.39/1.280	30.25/11.625	27.746/5.078	14.27/4.988	14.65/2.861	15.33/2.385
Average	51.11/0.033	33.78/0.037	44.95/2.178	21.68/0.710	29.39/6.987	20.943/5.080	14.16/2.806	14.68/1.603	14.90/1.370

Table 9: **Detailed results on the Brandimarte FJSP instances.** Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. CP-SAT$_{1,800\,s}$ is reported as Gap (mean makespan), where its solutions define the instance-wise reference makespans. The ten instances comprise two instances each of sizes 10×6, 15×8, and 20×10, and one instance each of sizes 10×15, 15×4, 20×5, and 20×15.

Size	Rule-based			Greedy learning-based			Solver-based	MSLPM (ours)		
	FIFO-SPT	MOR-EET	MWKR-SPT	DANIEL-G	DOAGNN-G	ReSCHED	CP-SAT$_{1,800s}$	$H=1$	$H=3$	$H=5$
10×6	41.79/0.03	34.67/0.03	38.17/0.03	13.37/0.26	32.60/0.27	8.27/0.30	0.00 (33.00)	10.66/0.18	22.37/0.08	10.86/0.06
10×15	61.56/0.09	57.50/0.10	58.44/0.10	26.56/0.73	65.62/0.77	14.06/1.07	0.00 (64.00)	0.76/0.48	3.87/0.21	3.62/0.16
15×4	8.67/0.06	8.79/0.06	9.13/0.06	2.31/0.50	8.09/0.50	2.89/0.63	0.00 (173.00)	2.59/0.32	3.92/0.14	4.45/0.11
15×8	16.59/0.07	14.14/0.07	11.89/0.08	10.83/0.59	21.08/0.57	10.00/0.82	0.00 (132.00)	7.15/0.28	7.76/0.13	9.03/0.09
20×5	49.08/0.06	47.52/0.06	48.94/0.06	14.18/0.46	51.06/0.50	4.96/0.59	0.00 (141.00)	4.01/0.08	5.28/0.04	5.58/0.03
20×10	10.14/0.16	6.43/0.16	7.60/0.16	2.44/1.20	10.10/1.43	3.58/1.99	0.00 (415.00)	3.98/0.88	3.56/0.40	3.07/0.31
20×15	23.60/0.17	19.37/0.17	19.73/0.17	7.21/1.30	20.27/1.57	0.90/2.24	0.00 (222.00)	3.79/0.97	4.05/0.45	3.58/0.35
Average	**27.99/0.12**	**24.36/0.12**	**25.16/0.12**	**10.35/0.89**	**27.26/1.00**	**6.65/1.42**	**0.00 (176.00)**	**5.47/0.45**	**8.45/0.21**	**6.32/0.16**

Table 10: **Detailed results on the Hurink `edata` FJSP instances.** Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. CP-SAT$_{1,800\,\mathrm{s}}$ is reported as Gap (mean makespan), where its solutions define the instance-wise reference makespans. Each size group contains five instances.

Size	Rule-based			Greedy learning-based			Solver-based	MSLPM (ours)		
	FIFO-SPT	MOR-EET	MWKR-SPT	DANIEL-G	DOAGNN-G	ReSCHED	CP-SAT$_{1,800\,\mathrm{s}}$	$H=1$	$H=3$	$H=5$
10×5	18.49/0.16	16.14/0.16	19.98/0.16	15.92/0.45	14.74/0.20	18.00/0.26	0.00 (577.00)	7.95/0.04	12.92/0.02	12.92/0.01
10×10	23.32/0.33	23.03/0.33	19.56/0.33	19.60/0.89	19.08/0.49	13.25/0.63	0.00 (818.80)	10.85/0.09	10.85/0.04	10.50/0.03
15×5	12.64/0.24	13.04/0.24	14.46/0.24	14.11/0.68	10.12/0.32	11.32/0.41	0.00 (836.80)	9.24/0.06	14.29/0.03	11.16/0.02
15×10	27.00/0.51	25.47/0.51	20.25/0.51	19.17/1.35	17.89/0.77	21.70/1.08	0.00 (937.80)	14.85/0.15	18.27/0.07	19.02/0.06
15×15	25.17/0.83	24.92/0.83	22.44/0.83	19.48/2.09	18.18/1.41	16.32/2.01	0.00 (1205.60)	23.25/0.28	14.25/0.14	14.07/0.11
20×5	14.60/0.33	11.82/0.33	12.26/0.33	10.18/0.89	13.78/0.49	13.52/0.56	0.00 (1070.00)	13.95/0.09	14.22/0.04	15.39/0.03
20×10	26.95/0.69	23.93/0.70	24.05/0.70	18.46/1.83	20.53/1.17	18.65/1.55	0.00 (1154.80)	18.03/0.24	15.80/0.12	14.56/0.09
30×10	19.11/1.09	16.22/1.09	16.42/1.09	14.34/2.80	15.43/2.22	13.79/2.98	0.00 (1624.80)	10.25/0.44	18.66/0.22	8.29/0.18
Average	**20.91/0.52**	**19.32/0.52**	**18.68/0.52**	**16.41/1.37**	**16.22/0.88**	**15.82/1.18**	**0.00 (1028.20)**	**13.55/0.18**	**14.91/0.09**	**13.24/0.07**

Table 11: **Detailed results on the Hurink `rdata` FJSP instances.** Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. CP-SAT$_{1,800\,\mathrm{s}}$ is reported as Gap (mean makespan), where its solutions define the instance-wise reference makespans. Each size group contains five instances.

Size	Rule-based			Greedy learning-based			Solver-based	MSLPM (ours)		
	FIFO-SPT	MOR-EET	MWKR-SPT	DANIEL-G	DOAGNN-G	ReSCHED	CP-SAT$_{1,800\,\mathrm{s}}$	$H=1$	$H=3$	$H=5$
10×5	15.32/0.16	13.28/0.16	17.27/0.16	11.47/0.45	8.78/0.19	9.81/0.25	0.00 (507.40)	5.91/0.04	6.13/0.02	4.16/0.01
10×10	24.02/0.33	23.26/0.33	20.81/0.33	21.65/0.90	19.73/0.48	14.14/0.62	0.00 (697.00)	14.42/0.09	16.01/0.04	8.86/0.03
15×5	9.05/0.24	7.77/0.24	5.89/0.24	4.70/0.67	5.96/0.32	6.64/0.41	0.00 (794.00)	6.51/0.06	2.38/0.03	8.26/0.02
15×10	25.23/0.51	21.11/0.51	18.84/0.51	14.37/1.36	13.49/0.76	12.93/1.06	0.00 (830.80)	12.64/0.15	24.41/0.07	14.75/0.06
15×15	22.24/0.83	19.12/0.83	19.18/0.83	17.92/2.07	17.03/1.42	14.32/2.02	0.00 (1033.60)	16.77/0.29	19.29/0.14	21.72/0.11
20×5	9.20/0.33	8.09/0.33	5.24/0.33	3.63/0.90	4.43/0.48	6.80/0.60	0.00 (1040.80)	3.12/0.08	2.39/0.04	3.31/0.03
20×10	16.49/0.70	13.96/0.70	10.71/0.70	7.12/1.82	10.41/1.19	9.26/1.59	0.00 (1075.00)	12.53/0.24	10.06/0.12	8.05/0.09
30×10	8.64/1.09	6.31/1.10	5.43/1.09	3.20/2.76	6.37/2.21	4.00/3.01	0.00 (1557.80)	5.59/0.45	5.21/0.23	3.22/0.18
Average	**16.27/0.52**	**14.11/0.53**	**12.92/0.52**	**10.51/1.37**	**10.77/0.88**	**9.74/1.20**	**0.00 (942.05)**	**9.69/0.18**	**10.73/0.09**	**9.04/0.07**

Table 12: **Detailed results on the Hurink `vdata` FJSP instances.** Each entry reports the mean optimality gap (%) and mean runtime (s) as Gap/Time. CP-SAT$_{1,800\,\mathrm{s}}$ is reported as Gap (mean makespan), where its solutions define the instance-wise reference makespans. Each size group contains five instances.

Size	Rule-based			Greedy learning-based			Solver-based	MSLPM (ours)		
	FIFO-SPT	MOR-EET	MWKR-SPT	DANIEL-G	DOAGNN-G	ReSCHED	CP-SAT$_{1,800\,\mathrm{s}}$	$H=1$	$H=3$	$H=5$
10×5	13.65/0.16	10.27/0.16	9.61/0.16	6.48/0.45	5.99/0.19	5.56/0.25	0.00 (507.20)	5.47/0.04	5.03/0.02	5.69/0.01
10×10	4.62/0.33	4.97/0.33	4.18/0.33	6.10/0.91	1.47/0.47	0.12/0.64	0.00 (684.80)	0.00/0.09	1.46/0.04	0.00/0.03
15×5	7.15/0.24	5.53/0.24	3.44/0.24	2.31/0.68	5.25/0.35	5.91/0.41	0.00 (794.20)	5.38/0.06	4.01/0.03	1.13/0.02
15×10	8.75/0.51	5.48/0.51	2.86/0.51	0.05/1.36	4.27/0.79	1.24/1.10	0.00 (804.40)	5.56/0.16	-0.43/0.07	1.67/0.06
15×15	0.02/0.83	-0.80/0.83	-0.77/0.83	-1.53/2.07	-0.70/1.43	-3.81/2.06	0.00 (995.40)	1.01/0.28	-0.26/0.14	0.47/0.11
20×5	4.71/0.32	3.83/0.33	2.07/0.33	1.19/0.91	3.44/0.52	3.49/0.60	0.00 (1040.80)	1.01/0.09	0.64/0.04	1.74/0.03
20×10	6.83/0.70	3.05/0.70	1.02/0.70	0.52/1.82	2.10/1.20	2.64/1.60	0.00 (1068.80)	3.05/0.24	4.78/0.12	2.90/0.09
30×10	3.94/1.09	2.36/1.10	0.71/1.09	0.67/2.77	2.27/2.21	1.39/3.02	0.00 (1558.40)	1.01/0.45	1.88/0.22	1.37/0.18
Average	**6.21/0.52**	**4.34/0.52**	**2.89/0.52**	**1.97/1.37**	**3.01/0.89**	**2.07/1.21**	**0.00 (931.75)**	**2.81/0.18**	**2.14/0.09**	**1.87/0.07**