SCHEDULERFM: REWARD-FREE PRETRAINING FOR GENERALIZABLE JOB-SHOP SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent studies using deep reinforcement learning (RL) for job-shop scheduling problems (JSSP) have largely relied on task-specific training, where separate policies are optimized for fixed scheduling objectives and settings. This limits reuse when the instance distribution, scale, objective, or scheduling setting changes. We propose *SchedulerFM*, a strategy foundation model for JSSP that decouples learning scheduling behavior from downstream optimization. SchedulerFM comprises two stages. First, *reward-free RL pretraining* learns a reusable latent strategy space by factorizing the long-horizon successor occupancies induced by diverse scheduling trajectories, without using a downstream scheduling reward. Second, *latent-space test-time optimization* freezes the pretrained model and searches over the strategy space, evaluating each candidate based on the downstream objective achieved by its complete rollout. A single JSSP-pretrained model transfers zero-shot across unseen distributions and scales, to flexible JSSP, and to previously unseen scheduling objectives, matching or outperforming strong learning and search baselines without additional training. More broadly, SchedulerFM opens a path toward reusable neural schedulers whose solution quality can scale with test-time compute without repeated task-specific retraining.

1 INTRODUCTION

Job-shop scheduling problems (JSSPs) require coordinating operations across shared machines under precedence and resource constraints (Błażewicz et al., 2000; Xiong et al., 2022). Learning-based schedulers (Zhang et al., 2020; Park et al., 2021b; Falkner et al., 2022; Tassel et al., 2023; Chalumeau et al., 2023; Zhang et al., 2024a;b; Ho et al., 2024; Liao et al., 2025) provide a promising way to amortize the computational effort of solving such problems by learning decision rules from data. In practical deployment, however, schedulers also need to operate beyond the settings encountered during training, *e.g.,* on substantially larger instances, different machine configurations, or under new operational objectives. This motivates methods that can retain the efficiency of learned scheduling while allowing behavior to be reused and adapted across problem settings.

Recent advances in neural combinatorial optimization (NCO) have substantially improved learning-based approaches to JSSP, including through structured representations based on disjunctive graphs and graph neural networks. Building on this progress, several challenges remain important to address for broader deployment: (1) *Generalization across scales and distributions*: A policy trained on a particular family of instances can encounter substantially different problem sizes or distributions at test time. (2) *Adaptation across objectives*: Many learning-based JSSP methods are trained with a specified reward or objective, whereas practical scheduling systems also need to support multiple or changing objectives. (3) *Effective use of test-time computation*: When additional computation is available at deployment, it is desirable to use it to adapt the scheduler to the particular instance and objective rather than relying solely on a fixed policy. These considerations motivate separating the learning of reusable scheduling behavior from the optimization of downstream objectives.

We investigate this separation through reward-free pretraining for JSSP, with the goal of adapting a shared pretrained model across instance scales, related scheduling settings, and downstream objectives. These dimensions of reuse allow us to assess whether pretraining captures transferable scheduling strategies beyond a single training objective. Specifically, we ask:

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.

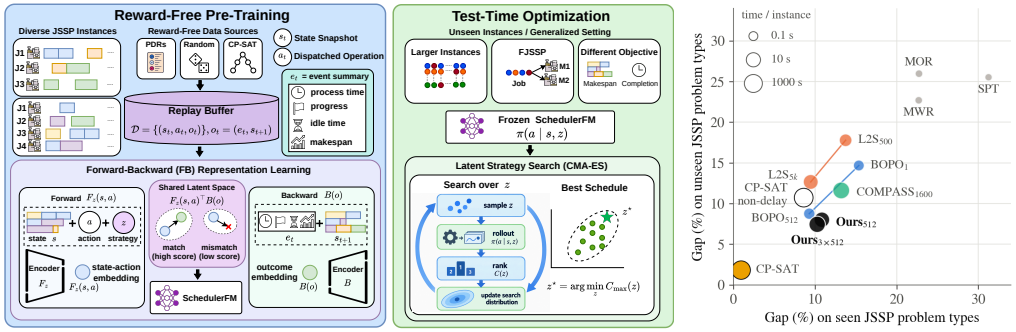


Figure 1: **Overview of SchedulerFM. Left:** Reward-Free RL Pretraining learns a reusable latent strategy space from diverse scheduling trajectories through FB representations. **Middle:** The frozen model adapts to new instances, scheduling settings, and objectives through latent-space test-time optimization. **Right:** quality-compute trade-off across seen and unseen JSSP problem types; marker size denotes time per instance, and additional test-time search further improves SchedulerFM.

> *Can reward-free pretraining learn a shared repertoire of scheduling strategies that supports zero-shot generalization across scales, objectives, and scheduling settings?*

We answer this question in the affirmative by proposing **SchedulerFM**, a strategy foundation model for JSSP that decouples learning reusable scheduling behavior from optimizing a particular downstream objective. SchedulerFM consists of two stages: (i) **Reward-Free RL Pretraining** learns a reusable latent space of long-horizon scheduling strategies from offline scheduling transitions using forward–backward (FB) representations of successor measures, without optimizing a downstream scheduling reward; (ii) **Latent-Space Test-Time Optimization** freezes the pretrained model and searches this strategy space for each test instance and objective. At test time, reward-derived anchors provide informed starting points for CMA-ES (Hansen & Ostermeier, 2001). Each candidate strategy induces a complete rollout, whose resulting schedule is evaluated directly by the downstream objective. The search distribution is then updated toward better strategies. Thus, adaptation requires no parameter updates, and additional test-time compute can be spent directly on finding better schedules. Figure 1 illustrates this two-stage framework and summarizes its quality–compute trade-off: the same frozen model remains effective on unseen JSSP problem types and can convert additional test-time compute into better solutions.

On JSSP benchmarks, SchedulerFM achieves the smallest average gap in makespan compared to all the benchmark learning methods. On extreme scales up to 1000×100, it achieves comparable or better gaps than the best-performing method across all tested scales. The same checkpoint transfers zero-shot to flexible JSSP (FJSSP), achieving a 4.28% average gap compared with 4.38% for the strongest compared learning baseline, despite receiving no FJSSP pretraining or fine-tuning. It further outperforms all compared heuristic and latent-search baselines on all four unseen scheduling objectives without updating the pretrained model. Together, these demonstrate that SchedulerFM learns reusable scheduling strategies rather than a policy specialized to a particular training regime.

Our main contributions are three-fold: (1) We introduce **reward-free pretraining for neural scheduling**, using forward-backward representations to learn a reusable latent space of long-horizon scheduling strategies from offline reward-free transitions. (2) We propose **test-time latent strategy search**, which adapts a frozen scheduler to each instance and objective through exact rollout evaluation, enabling additional test-time compute to improve solutions without retraining. (3) Through extensive experiments, we demonstrate favorable **zero-shot generalization** across unseen JSSP distributions and scales, previously unseen scheduling objectives, and zero-shot transfer from JSSP to FJSSP using a single pretrained checkpoint.

2 Related Work

A prominent line of work addresses JSSP with *constructive* methods, which learn to construct schedules sequentially through dispatching policies, often with graph-based representations. Represen-

tative methods include the seminal L2D (Zhang et al., 2020) and RL-GNN (Park et al., 2021b), as well as approaches based on constraint programming (Tassel et al., 2023), curriculum learning (Iklassov et al., 2023), and residual scheduling (Ho et al., 2024). In contrast, *improvement-based* methods initialize a complete schedule and iteratively refine it through local modifications: NeuroLS (Falkner et al., 2022) uses RL to select local-search operators adaptively, L2S (Zhang et al., 2024a) combines deep RL with an improvement heuristic, and TBGAT (Zhang et al., 2024b) further incorporates topological representation learning and bidirectional graph attention into the improvement pipeline. Other work explores alternative learning and adaptation paradigms. Self-labeling (Corsini et al., 2024) uses locally optimized solutions as pseudo-labels for supervised training, whereas BOPO (Liao et al., 2025) constructs preference pairs from sampled solutions and employs objective-guided preference optimization to improve sample efficiency, with evaluations on both standard JSSP and flexible JSSP. COMPASS (Chalumeau et al., 2023) instead learns a latent-conditioned distribution of specialized policies during training and searches the latent space at inference time to adapt the policy to individual instances.

Parallel efforts have advanced scheduling representations and architectures. ScheduleNet frames scheduling as a multi-agent problem (Park et al., 2021a), while DAG-based models explicitly capture precedence and resource constraints (Jeon et al., 2023). Moreover, RL has been extended from classical JSSP to FJSP, where policies must jointly determine operation sequences and machine assignments (Song et al., 2023; Yuan et al., 2024; Liu et al., 2025; Li et al., 2025). Collectively, these studies demonstrate the effectiveness of learned policies across constructive, improvement-based, preference-based, adaptation-based, and rolling-horizon paradigms. However, most existing approaches are trained for a particular objective or scheduling setting, while comparatively little attention has been given to whether a scheduling policy can acquire broadly reusable knowledge through pretraining and subsequently transfer to different objectives and new scheduling settings in a completely zero-shot manner at test time.

Separately, broader research on neural combinatorial optimization has investigated policy adaptation, learned local search, preference optimization, and cross-task transfer (Falkner et al., 2022; Chalumeau et al., 2023; Liao et al., 2025; Luo et al., 2025; Fang et al., 2026; Yu et al., 2026). Although these methods offer relevant mechanisms and empirical evidence across routing and scheduling problems, they do not study reward-free RL pretraining for acquiring transferable scheduling knowledge. To the best of our knowledge, SchedulerFM is the first systematic study of RL pretraining for neural combinatorial optimization, with job-shop scheduling as its primary testbed. Due to the page limit, additional related work is discussed in Appendix C.

3 PRELIMINARIES

Job-Shop Scheduling Problem. A standard JSSP instance of size $n \times m$ consists of n jobs $\mathcal{J}$ and m machines $\mathcal{M}$. Each job $j \in \mathcal{J}$ comprises an ordered sequence of operations $O_{j,1} \to \cdots \to O_{j,m}$. Each operation $O_{j,k}$ has a processing time $p_{j,k} > 0$ and must be processed non-preemptively on its prescribed machine $\mu(j,k) \in \mathcal{M}$. Operations belonging to the same job must satisfy their precedence constraints, while each machine can process at most one operation at a time.

Let C_j denote the completion time of job j. Following the standard JSSP literature, we primarily consider *makespan minimization*, with $C_{\max} := \max_{j \in \mathcal{J}} C_j$ as the main objective. Given a due date d_j and a weight $\omega_j > 0$ for each job, and the tardiness $T_j := \max\{C_j - d_j, 0\}$, we also consider *total flow time* $\sum_{j \in \mathcal{J}} C_j$, *total tardiness* $\sum_{j \in \mathcal{J}} T_j$, *weighted tardiness* $\sum_{j \in \mathcal{J}} \omega_j T_j$ and *maximum tardiness* $\max_{j \in \mathcal{J}} T_j$. All jobs are released at time 0.

Reward-Free RL and Forward-Backward Representations. We consider an episodic *reward-free* Markov decision process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, state-dependent feasible action sets $\mathcal{A}(s) \subseteq \mathcal{A}$, transition kernel $P(\cdot \mid s, a)$, and discount factor $\gamma \in (0, 1]$. A policy $\pi(\cdot \mid s)$ specifies a conditional distribution over feasible actions $\mathcal{A}(s)$ at each state $s \in \mathcal{S}$.

In standard RL, a policy is typically learned for a particular reward function. Reward-free RL instead separates *representation learning* from *reward specification*: a representation of the environment's future behavior is pretrained from interactions without committing to a specific downstream reward (Touati & Ollivier, 2021; Touati et al., 2023). At deployment time, a new reward function

can then be mapped into this pretrained representation to obtain a corresponding policy, without updating the pretrained model parameters.

- **Successor Occupancy**: The key object underlying reward-free RL is a representation of what outcomes are likely to be encountered in the future. Let $\mathcal{X}$ denote a space of future outcomes of interest, and let $X_t \in \mathcal{X}$ denote the outcome observed at step t. For a policy π, define the *discounted successor measure* over $\mathcal{X}$ by

$$M^\pi(s, a, \mathcal{E}) := (1-\gamma) \cdot \mathbb{E}_\pi \Big[\sum\nolimits_{t \geq 0} \gamma^t \mathbf{1}\{X_t \in \mathcal{E}\} \,|\, s_0 = s,\ a_0 = a \Big], \tag{1}$$

  where $\mathcal{E}$ is a measurable subset of $\mathcal{X}$. Intuitively, $M^\pi(s, a, \mathcal{E})$ measures the discounted amount of future occupancy that policy π assigns to outcomes in $\mathcal{E}$ after taking action a in state s. Let ρ be a reference distribution over $\mathcal{X}$ whose support covers the outcomes of interest. Assuming that $M^\pi(s, a, \cdot)$ is absolutely continuous with respect to ρ, we denote its density by $m^\pi(s, a, x) := \frac{dM^\pi(s,a,\cdot)}{d\rho}(x)$. We refer to this generalized successor-measure formulation over $\mathcal{X}$ as *successor occupancy* and call $m^\pi(s, a, x)$ the corresponding *successor occupancy density*. This representation is useful because it separates future dynamics from the reward function. For a reward function $r : \mathcal{X} \to \mathbb{R}$, the corresponding action value can be written as

$$Q_r^\pi(s, a) = \mathbb{E}_{x \sim \rho} \left[m^\pi(s, a, x)\, r(x) \right]. \tag{2}$$

  In standard reward-free RL formulations, x is typically either a future state or a future state-action pair. SchedulerFM instead instantiates x as a scheduling transition outcome in the context of JSSP.

- **Forward-Backward Representations**: Directly representing the successor occupancy $m^\pi(s, a, x)$ for all policies and outcomes can be prohibitively high-dimensional. Forward-backward (FB) representations (Touati & Ollivier, 2021) address this by learning a low-dimensional factorization. Let $z \in \mathbb{R}^d$ denote a latent task vector, where d is the latent dimension. Let $\mathcal{Z}$ denote the set of all possible latent vectors $\boldsymbol{z}$. FB considers a family of policies $\{\pi_{\boldsymbol{z}}\}_{\boldsymbol{z} \in Z}$ and models their successor occupancies using a latent-conditioned forward map $\mathbf{F}_{\boldsymbol{z}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and a shared backward map $\mathbf{B} : \mathcal{X} \to \mathbb{R}^d$. Then, the successor occupancy is approximated by

$$m^{\pi_{\boldsymbol{z}}}(s, a, x) \approx \mathbf{F}_{\boldsymbol{z}}(s, a)^\top \mathbf{B}(x), \qquad \pi_{\boldsymbol{z}}(s) := \operatorname*{arg\,max}_{a \in \mathcal{A}(s)} \mathbf{F}_{\boldsymbol{z}}(s, a)^\top \boldsymbol{z}. \tag{3}$$

  Given a downstream reward function r, FB embeds the reward into this latent space as

$$\boldsymbol{z}_r := \mathbb{E}_{x \sim \rho} \left[r(x) \mathbf{B}(x) \right]. \tag{4}$$

  Combining this reward embedding with the FB factorization yields

$$Q_r^{\pi_{\boldsymbol{z}}}(s, a) \approx \mathbf{F}_{\boldsymbol{z}}(s, a)^\top \boldsymbol{z}_r. \tag{5}$$

  In this manner, FB pretraining learns a shared representation of future behavior, while different downstream reward functions correspond to different latent directions $\boldsymbol{z}_r$.

4 Methodology

4.1 Reward-Free Constructive Scheduling MDP

We instantiate the reward-free MDP from Section 3 as a constructive scheduling process under the classical non-delay scheme (Baker, 1974; Pinedo, 2016). A complete schedule is constructed by repeatedly dispatching one eligible operation, without specifying a task-dependent scalar reward.

State and Action. At decision step t, the state $s_t = \big((k_j, \eta_j)_{j=1}^n, (\alpha_c)_{c=1}^m\big)$ encodes the residual scheduling problem, where k_j is the index of job j's next unscheduled operation, η_j is the earliest job-feasible start time, and α_c is the availability time of machine $c \in \mathcal{M}$. The next decision time is $T(s_t) = \min_{j:k_j \leq m} \max\{\eta_j, \alpha_{\mu(j,k_j)}\}$, with the set of feasible actions as

$$\mathcal{A}(s_t) = \left\{ O_{j,k_j} \;\middle|\; k_j \leq m,\ \max\{\eta_j, \alpha_{\mu(j,k_j)}\} \leq T(s_t) \right\}. \tag{6}$$

An action $a_t \in \mathcal{A}(s_t)$ dispatches one eligible operation.

Transition Outcome. Dispatching O_{j,k_j} starts at $\max\{\eta_j, \alpha_{\mu(j,k_j)}\}$ and updates the corresponding job and machine availability times after processing for p_{j,k_j}. Transitions are deterministic, and each episode terminates after exactly nm dispatching decisions, once all operations have been scheduled.

To characterize the consequence of each dispatch, we augment the successor state with an event vector $e_t \in \mathbb{R}^7$ containing (i) the processing time, (ii) makespan increment, (iii) machine-idle increment, (iv) job completion time, (v) workload-spread change, (vi) progress, and (vii) terminal indicator. We define the transition outcome at step t as $o_t := (e_t, s_{t+1})$. Notably, these statistics describe the outcome of a dispatch without combining them into a predefined reward. The non-delay restriction substantially reduces the constructive search space, although it does not generally contain every optimal JSSP schedule (Pinedo, 2016).

4.2 Outcome-Based FB Strategy Model

SchedulerFM specializes the generic FB variable x of Section 3 to the scheduling outcome $o = (e, s')$. Thus, the backward map represents future scheduling outcomes rather than future state-action pairs. We constrain latent vectors to $\mathcal{Z} := \{\boldsymbol{z} \in \mathbb{R}^d : \|\boldsymbol{z}\|_2 = \sqrt{d}\}$. Although $\boldsymbol{z}$ is a task vector in standard FB terminology, we refer to it as a *latent strategy*, since fixing $\boldsymbol{z}$ selects one scheduling behavior. SchedulerFM uses twin forward maps $\mathbf{F}^{(u)}_{\boldsymbol{z}}(s,a) \in \mathbb{R}^d$, $u \in \{1,2\}$, and a shared backward map $\mathbf{B}(o) \in \mathbb{R}^d$. The backward representation is normalized onto $\mathcal{Z}$. We learn the advantage-form FB approximation

$$m^{\pi_{\boldsymbol{z}}}(s,a,o) \approx \mathbf{F}^{(u)}_{\boldsymbol{z}}(s,a)^\top \mathbf{B}(o) + b(s,\boldsymbol{z},e), \qquad u \in \{1,2\}, \tag{7}$$

where $b(s,\boldsymbol{z},e)$ is an action-independent baseline. $\mathbf{F}^{(u)}_{\boldsymbol{z}}(s,a)^\top \mathbf{B}(o)$ carries the action-dependent component of future outcome occupancy. Then, the corresponding dispatching policy is

$$\pi_{\boldsymbol{z}}(s) = \arg\max_{a \in \mathcal{A}(s)} \min_{u \in \{1,2\}} \mathbf{F}^{(u)}_{\boldsymbol{z}}(s,a)^\top \boldsymbol{z}, \tag{8}$$

where the min operator is motivated by the TD3-style learning of twin critics (Fujimoto et al., 2018) for mitigating overestimation. The same $\boldsymbol{z}$ is held fixed throughout an episode, so each latent strategy induces one complete non-delay schedule.

Size-Agnostic Graph Representation. To generalize across instance sizes, SchedulerFM represents each residual problem as a heterogeneous graph of remaining operations and machines. The forward map uses a graph-attention encoder (Veličković et al., 2018) over job and machine relations, while the backward map uses a heterogeneous GIN (Xu et al., 2019) to encode the successor graph together with the event e and global scheduling statistics. Permutation-invariant aggregation yields fixed-dimensional representations, allowing the same parameters to be applied to different numbers of jobs and machines. Full architecture and training configurations are provided in Appendix A.1.

4.3 Reward-Free RL Pretraining

SchedulerFM is trained entirely offline from a fixed replay buffer $\mathcal{D} = \{(s_i, a_i, o_i)\}_{i=1}^N$. The buffer combines trajectories generated by priority dispatching rules, uniform-random dispatching, and high-quality CP-SAT solutions projected into the same non-delay action space. This mixture exposes the model to both diverse scheduling behaviors and high-quality decisions. Importantly, no downstream reward is attached to these transitions. Instead, pretraining learns a reusable representation of how scheduling decisions influence future outcomes, which can later be adapted to different objectives through the latent strategy vector $\boldsymbol{z}$. The complete replay-buffer construction procedure is given in Appendix A.2.

Latent Strategy Sampling. During pretraining, each latent vector $\boldsymbol{z}$ defines a different direction in the learned space of scheduling behaviors. To expose the model to a broad and meaningful set of such directions, for each transition i in a minibatch, we sample $\boldsymbol{z}_i$ with equal probability from either a generic latent prior or the backward representation of an observed outcome:

$$\boldsymbol{z}_i = \begin{cases} \boldsymbol{z}, \quad \boldsymbol{z} \sim \mathrm{Unif}(\mathcal{Z}), & \text{with probability } \frac{1}{2}, \\ \mathrm{sg}\big(\mathbf{B}(o_{\sigma(i)})\big), & \text{with probability } \frac{1}{2}, \end{cases} \tag{9}$$

where σ is a random permutation of the minibatch and $\mathrm{sg}(\cdot)$ denotes the stop-gradient operator. Uniform samples encourage broad coverage of the latent space, whereas outcome-derived samples anchor latent directions to scheduling behaviors observed in the replay buffer.

FB Training Objective. The FB objective trains the model to predict the discounted occupancy of future scheduling outcomes under each latent-conditioned policy. Specifically, for minibatch transitions (s_i, a_i, o_i) and outcomes $o_j = (e_j, s'_j)$, we define

$$\widehat{m}_{ij}^{(u)} := \mathbf{F}_{z_i}^{(u)}(s_i, a_i)^\top \mathbf{B}(o_j) + b(s_i, \boldsymbol{z}_i, e_j), \tag{10}$$

where $u \in \{1, 2\}$ indexes the two forward maps. Let $\bar{m}_{ij}$ denote the one-step bootstrapped FB target, which is computed using Polyak-averaged target networks and the latent-conditioned Boltzmann policy at the successor state s'_i. The FB target is zero for terminal transitions. We optimize

$$\mathcal{L}_{\mathrm{FB}}(\mathbf{F}, \mathbf{B}) = \underbrace{\frac{1}{2} \sum_{u=1}^{2} \mathbb{E}_{i \neq j} \left[\left(\widehat{m}_{ij}^{(u)} - \bar{m}_{ij} \right)^2 \right]}_{(a)} - \underbrace{\sum_{u=1}^{2} \mathbb{E}_i \left[\widehat{m}_{ii}^{(u)} \right]}_{(b)}. \tag{11}$$

The off-diagonal term (a) enforces consistency between predicted and bootstrapped future occupancies, while the diagonal term (b) captures the immediate occupancy of the outcome actually observed in the transition. Together, these terms train SchedulerFM to model future scheduling outcomes without requiring a predefined task reward. The detailed description of the bootstrapped FB target and auxiliary FB training objectives are provided in Appendix A.3.

4.4 Latent-Space Test-Time Optimization

After pretraining, all SchedulerFM network parameters are frozen. Rather than retraining the model for each downstream objective or test instance, we adapt only the low-dimensional latent $\boldsymbol{z}$. Intuitively, pretraining provides a family of scheduling behaviors indexed by $\boldsymbol{z}$, and test-time optimization searches this latent space for the behavior that performs best on the current instance.

Reward-Derived Anchors. We first construct informed starting points for latent-space search using the FB reward embedding introduced in Equation (4). Four event coordinates correspond to additive scheduling costs: (i) makespan increment, (ii) machine-idle increment, (iii) job completion time, and (iv) workload-spread change. Let $\boldsymbol{\phi}(o) \in \mathbb{R}^4$ collect the negatives of these quantities so that larger values correspond to lower costs. For any linear event reward $r_{\boldsymbol{w}}(o) = \boldsymbol{w}^\top \boldsymbol{\phi}(o)$ with $\boldsymbol{w} \in \mathbb{R}^4$, the FB reward embedding is

$$\boldsymbol{z}_{r_{\boldsymbol{w}}} = \boldsymbol{\Lambda} \boldsymbol{w}, \quad \boldsymbol{\Lambda} := \mathbb{E}_{o \sim \rho} \left[\mathbf{B}(o) \boldsymbol{\phi}(o)^\top \right]. \tag{12}$$

We estimate $\boldsymbol{\Lambda}$ once from the pretraining data and project $\boldsymbol{\Lambda} \boldsymbol{w}$ onto $\mathcal{Z}$ to obtain a *reward-derived anchor*. These anchors provide principled initial search directions associated with different scheduling preferences. However, because the learned FB representation is approximate and the downstream objective may not be exactly represented by a linear event reward, we use them only as search initializations rather than final solutions.

Search by Realized Schedule Quality. We ultimately evaluate a latent strategy by the quality of its produced complete schedule. Let $C(\boldsymbol{z})$ denote the downstream cost obtained from a complete greedy rollout of $\pi_{\boldsymbol{z}}$. For makespan minimization, $C(\boldsymbol{z}) = C_{\max}(\boldsymbol{z})$. Test-time adaptation solves

$$\boldsymbol{z}^\star = \arg\min_{\boldsymbol{z} \in \mathcal{Z}} C(\boldsymbol{z}). \tag{13}$$

Importantly, $C(\boldsymbol{z})$ is the realized schedule-level objective, rather than the FB score used during pretraining or the linear reward used to construct an anchor. Changing the downstream objective requires only changing the schedule-level $C(\boldsymbol{z})$, while the pretrained SchedulerFM remains fixed.

CMA-ES Search. The mapping $\boldsymbol{z} \mapsto C(\boldsymbol{z})$ involves a complete discrete scheduling rollout and is non-differentiable, and we therefore treat Equation (13) as a derivative-free black-box optimization problem. We use CMA-ES (Hansen & Ostermeier, 2001), with the initial candidate population combines reward-derived anchors with uniformly sampled latent strategies. Each candidate $\boldsymbol{z}$ is projected onto $\mathcal{Z}$, held fixed throughout one complete greedy rollout, and evaluated by its realized schedule cost. CMA-ES then updates its search distribution toward better-performing latent strategies until the rollout budget is exhausted. Full search configurations and comparisons with alternative black-box optimizers are provided in Appendix A.4 and Appendix B.3, respectively.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Datasets and Generalization Settings. We pretrain SchedulerFM only on synthetic JSSP instances with at most 30 jobs and 20 machines (Section 4.3) by following Taillard's construction (Taillard, 1993). We evaluate the pretrained model in four settings: (i) **JSSP benchmark generalization** evaluates the models on 170 public instances from four benchmark families, namely LA (Lawrence, 1984), SWV (Storer et al., 1992), Taillard (TA) (Taillard, 1993), and DMU (Demirkol et al., 1998). The evaluated instances span sizes from 10×5 to 100×20. The exact per-family composition is reported in Table 1. (ii) **Extreme-scale generalization** tests whether the same checkpoint remains reliable far beyond the pretraining scale. We use 100×100, 100×1000, and 1000×100 Taillard-distributed instance from Da Col & Teppan (2022), with 5 instances per shape. (iii) **Cross-problem transfer** tests whether behaviors learned on classical JSSP transfer to flexible JSSP (FJSSP), where machine selection becomes part of the scheduling decision. We evaluate on the Brandimarte benchmark (Brandimarte, 1993) and the edata, rdata, and vdata benchmarks of Hurink et al. (1994), *without* any FJSSP pretraining or fine-tuning. (iv) **Cross-objective transfer** tests whether a frozen reward-free representation remains useful when the downstream scheduling objective changes. Because alternative objectives do not have the same standardized best-known solution records as makespan, we evaluate 58 relatively small FT (Fisher & Thompson, 1963), LA, ABZ (Adams et al., 1988), and ORB (Applegate & Cook, 1991) instances, for which a one-hour unrestricted CP-SAT can provide a strong common reference. We consider total flow time, total tardiness, weighted tardiness, and maximum tardiness, with makespan included as a control.

Baselines. We compare against five groups of methods: (i) *Combinatorial optimization*: Google OR-Tools CP-SAT (Perron & Furnon, 2025), with a one-hour limit per instance, run both unrestricted and restricted to the non-delay schedule class used by SchedulerFM. (ii) *Greedy constructive*: three priority dispatching rules (Haupt, 1989), including shortest processing time (SPT), most operations remaining (MOR), and most work remaining (MWR), and the neural dispatchers L2D (Zhang et al., 2020), RL-GNN (Park et al., 2021b), and SLIM (Corsini et al., 2024). (iii) *Improvement*: L2S (Zhang et al., 2024a), evaluated with 500 and 5,000 improvement steps following the original paper. (iv) *Sampling constructive*: SLIM and BOPO (Liao et al., 2025), reporting the best of 512 sampled schedules,and DANIEL (Wang et al., 2024b) and ReSched (Xiao et al., 2026) for FJSSP. (v) *Latent search*: COMPASS (Chalumeau et al., 2023), using its original budget of 1,600 rollouts per instance. We rerun L2S, BOPO, COMPASS, DANIEL, and ReSched from their released code and checkpoints for fair comparisons.

Evaluation Protocol. We report the percentage gap $\mathrm{Gap}(\%) := 100(J - J^{\mathrm{ref}})/J^{\mathrm{ref}}$, where J is the objective value returned by a method and J^{ref} is a setting-specific reference (lower is better). For standard JSSP and FJSSP, the objective is makespan and J^{ref} is the instance best-known solution (BKS). For extreme-scale benchmarks, we use the published best-known upper bounds of Da Col & Teppan (2022). For cross-objective transfer, where standardized BKS values are unavailable, J^{ref} is the result of an unrestricted one-hour CP-SAT solve under the same objective. Further benchmark and evaluation details are provided in Appendix A.6.

Implementation Details. SchedulerFM is pretrained on 6,144 generated instances (512 for each of 12 training shapes), producing a replay buffer of 1.33M transitions. To evaluate the transferability of a single pretrained scheduler, we use the same fixed checkpoint for all benchmarks, objectives, and instance scales, without retraining, fine-tuning, or benchmark-specific checkpoint selection. Our default inference setting uses one latent-space search with 512 rollout evaluations per instance. Full training and inference configurations are provided in Appendix A.1. For FJSSP, the JSSP-pretrained model remains frozen, and we jointly optimize the latent strategy and a small machine-assignment parameterization with the same CMA-ES procedure, without FJSSP-specific training or parameter updates. Details are provided in Appendix A.5.

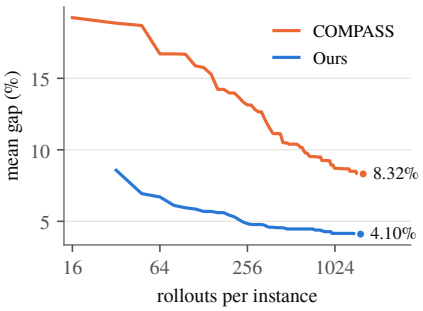
5.2 EXPERIMENTAL RESULTS

Can SchedulerFM Generalize Across JSSP Benchmarks? Table 1 evaluates the same frozen SchedulerFM checkpoint on 170 public JSSP instances. SchedulerFM achieves an average gap of 8.01%, compared with 8.76% for BOPO_{512}, 11.64% for COMPASS, and 12.66% for $\mathrm{L2S}_{5k}$. Across benchmark families, SchedulerFM performs best among the compared learning and search methods on SWV, TA, and DMU, while BOPO performs slightly better on LA.

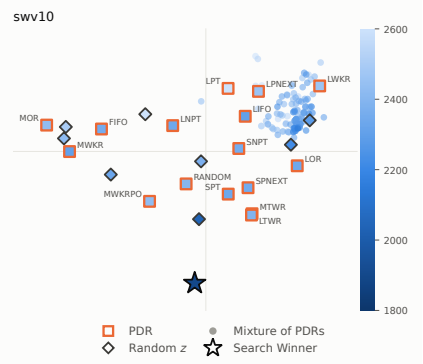
Table 1: **JSSP benchmarks.** Average makespan gap (%) to the BKS. "-" indicates unavailable results from the corresponding paper. Bold marks the best result excluding CP-SAT.

		CP-SAT (1 h)		Greedy Constructive						Improvement		Sampling Constructive		Latent Search	
Family	Size	Full	Non-delay	SPT	MOR	MWR	L2D	RL-GNN	SLIM	$L2S_{500}$	$L2S_{5k}$	$SLIM_{512}$	$BOPO_{512}$	COMPASS	SchedulerFM
LA	10×5	0.00	1.41	14.81	15.96	16.03	14.3	16.1	9.3	2.20	1.81	**1.1**	2.07	2.96	2.60
	15×5	0.00	0.00	14.86	3.93	5.49	5.5	1.1	2.6	**0.00**	**0.00**	**0.0**	**0.00**	**0.00**	**0.00**
	10×10	0.00	2.55	15.67	18.10	12.20	21.9	17.1	8.9	3.39	**1.98**	2.5	2.22	4.36	5.82
	20×5	0.00	0.00	13.72	3.79	5.17	4.2	2.1	2.1	**0.00**	**0.00**	**0.0**	**0.00**	0.13	**0.00**
	15×10	0.00	1.26	28.69	23.67	17.83	24.6	22.0	11.6	6.17	4.68	5.0	**4.59**	6.52	6.03
	20×10	0.00	1.19	33.43	20.87	17.23	24.7	27.3	12.1	7.96	**4.22**	5.6	5.25	8.14	6.30
	15×15	0.00	2.67	24.59	18.06	18.21	27.1	21.4	13.6	7.74	**5.30**	5.6	5.45	8.38	8.23
	30×10	0.00	0.00	13.89	6.50	8.66	8.4	6.3	2.0	1.00	**0.00**	**0.0**	**0.00**	0.54	**0.00**
	Avg	0.00	1.14	19.96	13.86	12.60	16.3	14.2	7.8	3.56	**2.25**	2.5	2.45	3.88	3.62
SWV	20×10	0.16	5.85	26.26	40.49	33.90	41.4	28.4	-	27.43	18.88	-	11.44	17.35	**11.05**
	20×15	2.47	13.74	32.05	40.87	33.20	40.6	29.4	-	27.15	17.80	-	13.10	18.51	**12.57**
	50×10	0.84	13.11	17.04	31.13	23.27	30.8	16.8	-	21.37	17.83	-	8.38	7.99	**4.49**
	Avg	1.08	11.45	23.10	35.91	28.41	35.9	22.9	-	24.33	18.08	-	10.32	12.96	**8.15**
TA	50×15	0.00	4.17	24.11	17.37	16.86	20.0	15.9	10.1	10.47	5.31	4.9	4.99	9.36	**4.53**
	50×20	0.59	8.43	25.54	17.68	17.95	23.9	21.3	11.6	13.24	**6.93**	7.6	7.20	11.26	7.36
	100×20	0.00	5.50	14.41	9.15	8.31	12.9	9.2	5.8	7.69	2.40	2.1	1.44	4.27	**1.06**
	Avg	0.20	6.03	21.35	14.73	14.37	18.9	15.5	9.2	10.47	4.88	4.9	4.54	8.30	**4.32**
DMU	20×15	1.31	10.20	28.27	30.26	28.59	39.0	-	18.0	21.65	14.24	11.3	**10.94**	15.75	11.53
	20×20	1.59	12.08	31.50	26.88	26.82	37.7	-	19.4	19.02	13.73	**12.3**	12.51	15.54	12.84
	30×15	2.98	16.94	31.96	36.42	31.94	42.0	-	21.8	26.41	18.07	14.0	13.31	18.39	**12.43**
	30×20	4.77	19.86	35.11	33.73	30.88	39.7	-	25.7	26.57	20.23	15.8	14.92	19.35	**14.71**
	40×15	3.41	15.38	24.01	35.63	26.87	35.6	-	17.5	26.09	19.93	10.9	10.95	14.60	**8.76**
	50×15	3.84	18.02	25.01	34.70	27.49	36.5	-	15.7	26.46	21.58	10.6	10.89	12.34	**7.25**
	40×20	4.77	22.00	37.28	36.12	32.30	39.6	-	22.2	29.52	22.03	14.8	15.54	18.28	**13.21**
	50×20	5.74	22.38	30.71	36.22	30.53	39.5	-	22.4	32.28	25.56	15.0	15.74	17.24	**11.77**
	Avg	3.55	17.11	30.48	33.74	29.43	38.7	-	20.3	26.00	19.42	13.1	13.10	16.43	**11.56**
All		1.83	10.73	25.53	25.97	22.69	29.6	-	-	17.78	12.66	-	8.76	11.64	**8.01**

These results show that a single pretrained latent strategy space transfers effectively across diverse benchmark families without benchmark-specific training or fine-tuning. Figure 2(a) shows that SchedulerFM continues to improve as more rollout evaluations are allocated at test time, and Figure 2(b) shows that the search winner need not coincide with any individual PDR anchor, suggesting that latent-space optimization finds instance-specific strategies beyond the hand-crafted rules. Additional scaling results, replay-buffer ablations, and latent-space visualizations are provided in Appendices B.1, B.2, and B.4.



(a) Test-time scaling on SWV 50 × 10.



(b) Visualized latent space on SWV10.

Figure 2: **(a)** More rollouts improve SchedulerFM. **(b)** Search explores beyond PDR anchors.

Does SchedulerFM Generalize to Extreme Scale? To assess how far the pretrained strategy space can extrapolate beyond its training regime, we evaluate extreme-scale instances containing up to 100,000 operations, compared with 600 in the largest pretraining instances. The two largest settings, 100 × 1000 and 1000 × 100, have opposite aspect ratios and probe distinct scaling regimes. As shown in Table 2, competing methods behave differently across these settings: CP-SAT achieves gaps of 3.70% and 22.10%, while $BOPO_{512}$ obtains 12.97% and 1.29%, respectively. SchedulerFM, by contrast, stays within 0.71% of the best competing result at every tested scale, even with rollout budgets reduced to 64 and 32 on the two largest instances. At this scale, the encoder observes only the next 20 operations of each job, matching the longest job seen during pretraining. (Appendix A.4)

Can JSSP Pretraining Transfer Zero-Shot to FJSSP? We test whether the learned strategy space transfers to FJSSP, where each operation additionally requires a machine-assignment decision among eligible machines. As shown in Table 3, the same frozen JSSP-pretrained SchedulerFM attains a 4.28% av-

Table 2: **Extremely large JSSP instances.** Average makespan gap (%) to the best-known upper bound on Taillard-distributed instances. Bold marks the best result in each row.

	FDD/MWKR	CP-SAT (1 h)	$L2S_{500}$	BOPO	$BOPO_{512}$	$SchedulerFM_{512/64/32}$
100×100	27.47	**12.19**	25.60	13.64	12.22	12.90
100×1000	2.77	3.70	**2.75**	15.37	12.97	3.30
1000×100	5.81	22.10	5.45	2.34	1.29	**1.22**

Table 3: **Zero-shot transfer from JSSP to FJSSP.** Average makespan gap (%) to the BKS on the public FJSSP benchmarks. SchedulerFM uses the frozen JSSP-pretrained checkpoint without FJSSP training, whereas the compared learning baselines use FJSSP-specific training. Bold marks the best result excluding CP-SAT.

		CP-SAT (1 h)	PDR			Sampling Constructive				Latent Search
Family	Set	Full	SPT	MOR	MWR	$DANIEL_{100}^{15\times10}$	$ReSched_{100}^{15\times10}$	$BOPO_{256}$	$BOPO_{512}$	SchedulerFM
Brandimarte	mk01–15	0.35	39.78	24.99	25.86	8.73	9.30	16.09	15.48	**4.74**
Hurink	edata	0.06	25.92	22.15	21.05	10.61	12.88	6.52	5.92	**5.85**
	rdata	0.22	28.24	18.55	16.87	7.19	7.65	4.59	**4.23**	6.02
	vdata	0.06	17.72	6.80	5.37	1.06	1.07	0.50	**0.44**	0.87
	Avg	0.11	23.96	15.83	14.43	6.29	7.20	3.87	**3.53**	4.25
All		0.13	25.09	16.49	15.25	6.46	7.35	4.74	4.38	**4.28**

Table 4: **Zero-shot transfer to unseen objectives.** Gap (%) of the total objective over all 58 instances to that of unrestricted one-hour CP-SAT. SchedulerFM is frozen; only the objective used to rank latent strategies changes. Bold marks the best result excluding CP-SAT.

	CP-SAT (1 h)		Greedy Constructive	Latent Search	
Objective	Full	Non-delay	Best of 22 PDRs	COMPASS	SchedulerFM
Makespan	0.00	1.52	10.80	4.98	**4.45**
Total flow time	0.00	4.52	8.64	12.90	**4.64**
Total tardiness	0.00	16.85	34.20	58.85	**18.08**
Weighted tardiness	0.00	24.96	54.61	101.27	**38.27**
Maximum tardiness	0.00	4.46	39.17	42.82	**18.13**

erage gap, compared with 4.38% for $BOPO_{512}$, which is trained specifically on FJSSP. On Brandimarte, SchedulerFM achieves 4.74% versus 15.48% for $BOPO_{512}$, while performance varies across the Hurink subsets. Overall, the results show that strategies learned without FJSSP training data remain competitive after introducing machine assignment, supporting transfer to the richer FJSSP setting. The zero-shot machine-assignment construction is in Appendix A.5.

Can SchedulerFM Adapt to Unseen Scheduling Objectives? We test whether the same pretrained representation can adapt to new downstream objectives. SchedulerFM and the calibration matrix $\mathbf{\Lambda}$ remain fixed, and only the realized schedule-level cost used to rank latent strategies changes. As shown in Table 4, SchedulerFM outperforms COMPASS and the best of 22 dispatching rules on all four unseen objectives. For weighted tardiness, it achieves a 38.27% gap, versus 54.61% for the best dispatching rule and 101.27% for COMPASS. Because test-time search is guided by realized schedule quality, the reward-derived anchors need not represent each objective exactly. These results support the central idea of reward-free pretraining: a fixed representation can be reused across objectives through test-time latent optimization.

6 Conclusion

We introduced SchedulerFM, a reward-free framework for learning reusable scheduling strategies rather than objective-specific policies. By combining *reward-free RL pretraining* with *latent-space test-time optimization*, SchedulerFM is a single frozen model that transfers across JSSP benchmarks, extreme-scale instances, FJSSP, and unseen objectives without parameter updates. Our results show that scheduling behavior can be pretrained once and adapted through test-time search, reducing repeated task-specific retraining. One remaining limitation is the cost of multiple complete rollouts during test-time adaptation; improving test-time efficiency is therefore an important direction for future work. Overall, SchedulerFM suggests a path toward general-purpose scheduling models that leverage test-time compute and transfer across broader optimization settings.

AI USE STATEMENT

We used LLMs for proofreading and polishing the manuscript, as well as for identifying and gathering relevant prior work. We reviewed and revised all LLM-assisted text and take full responsibility for the related work and all other content in this submission.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or safety-critical experiments. While the proposed method is evaluated only on standard scheduling benchmarks, real-world deployment should account for application-specific constraints and potential impacts on affected stakeholders.

REPRODUCIBILITY STATEMENT

We describe the architecture and training objectives in Section 4, and provide the training and inference configurations in Section 5.1 and Appendices A.1–A.4. Appendix A.5 details the application of SchedulerFM to FJSSP, while Appendix A.6 provides the benchmark and evaluation details. We will make the source code publicly available upon acceptance.

REFERENCES

Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

Kenneth R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York, 1974.

Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. Learning to handle complex constraints for vehicle routing problems. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:93479–93509, 2024.

Jacek Błażewicz, Erwin Pesch, and Małgorzata Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2): 317–331, 2000.

Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.

Felix Chalumeau, Shikha Surana, Clément Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and Tom Barrett. Combinatorial optimization with policy adaptation using latent space search. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:7947–7959, 2023.

Jinbiao Chen, Zizhen Zhang, Zhiguang Cao, Yaoxin Wu, Yining Ma, Te Ye, and Jiahai Wang. Neural multi-objective combinatorial optimization with diversity enhancement. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:39176–39188, 2023.

Jinbiao Chen, Zhiguang Cao, Jiahai Wang, Yaoxin Wu, Hanzhang Qin, Zizhen Zhang, and Yue-Jiao Gong. Rethinking neural multi-objective combinatorial optimization via neat weight embedding. In *International Conference on Learning Representations (ICLR)*, volume 2025, pp. 48510–48529, 2025a.

Jinbiao Chen, Jiahai Wang, Zhiguang Cao, and Yaoxin Wu. Neural multi-objective combinatorial optimization via graph-image multimodal fusion. In *International Conference on Learning Representations (ICLR)*, volume 2025, pp. 33964–33983, 2025b.

Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. *Advances in Neural Information Processing Systems (NeurIPS)*, 37: 105528–105551, 2024.

Giacomo Da Col and Erich C Teppan. Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives*, 9:100249, 2022.

Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:77416–77429, 2023.

Darko Drakulić, Sofia Michel, and Jean-Marc Andreoli. GOAL: A generalist combinatorial optimization agent learner. In *International Conference on Learning Representations (ICLR)*, volume 2025, pp. 52465–52488, 2025.

Jonas K Falkner, Daniela Thyssens, Ahmad Bdeir, and Lars Schmidt-Thieme. Learning to control local search for combinatorial optimization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 361–376. Springer, 2022.

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. INViT: A generalizable routing problem solver with invariant nested view transformer. In *International Conference on Machine Learning (ICML)*, pp. 12973–12992, 2024.

Zhanhong Fang, Debing Wang, Jinbiao Chen, Jiahai Wang, and Zizhen Zhang. UCPO: A universal constrained combinatorial optimization method via preference optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 36900–36908, 2026.

Henry Fisher and Gerald L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In John F. Muth and Gerald L. Thompson (eds.), *Industrial Scheduling*, pp. 225–251. Prentice-Hall, 1963.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pp. 1587–1596, 2018.

Yong Liang Goh, Zhiguang Cao, Yining Ma, Yanfei Dong, Mohammed Haroon Dupty, and Wee Sun Lee. Hierarchical neural constructive solver for real-world TSP scenarios. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 884–895, 2024.

Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant RL populations for combinatorial optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:48485–48509, 2023.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

Reinhard Haupt. A survey of priority rule-based scheduling. *Operations Research Spektrum*, 11(1): 3–16, 1989.

Kuo-Hao Ho, Jui-Yu Cheng, Ji-Han Wu, Fan Chiang, Yen-Chi Chen, Yuan-Yu Wu, and I-Chen Wu. Residual scheduling: A new reinforcement learning approach to solving job shop scheduling problem. *IEEE Access*, 12:14703–14718, 2024.

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

Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa De Retana, and Martin Takac. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 5350–5358, 2023.

Helga Ingimundardottir and Thomas Philip Runarsson. Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. *Journal of Scheduling*, 21(4): 413–428, 2018.

Wonseok Jeon, Mukul Gagrani, Burak Bartan, Weiliang Will Zeng, Harris Teague, Piero Zappi, and Christopher Lott. Neural DAG scheduling via one-shot priority sampling. In *International Conference on Learning Representations (ICLR)*, 2023.

Detian Kong, Yining Ma, Zhiguang Cao, Tianshu Yu, and Jianhua Xiao. Efficient neural collaborative search for pickup and delivery problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):11019–11034, 2024.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21188–21198, 2020.

S Lawrence. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. *GSIA, Carnegie Mellon University*, 1984.

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 (ICLR)*, 2025.

Zijun Liao, Jinbiao Chen, Debing Wang, Zizhen Zhang, and Jiahai Wang. BOPO: Neural combinatorial optimization via best-anchored and objective-guided preference optimization. In *International Conference on Machine Learning (ICML)*, pp. 37456–37475, 2025.

Xinning Liu, Li Han, Ling Kang, Jiannan Liu, and Huadong Miao. Preference learning based deep reinforcement learning for flexible job shop scheduling problem. *Complex & Intelligent Systems*, 11(2):144, 2025.

Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *International Conference on Learning Representations (ICLR)*, volume 2025, pp. 14754–14780, 2025.

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:11096–11107, 2021.

Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-Opt. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:49555–49578, 2023.

Anton Milan, S Rezatofighi, Ravi Garg, Anthony Dick, and Ian Reid. Data-driven approximations to NP-hard problems. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 31, 2017.

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. ScheduleNet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021a.

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, 2021b.

Laurent Perron and Vincent Furnon. OR-Tools v9.14. Google, 2025. URL https://developers.google.com/optimization/.

Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, Cham, 5th edition, 2016.

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

Robert H Storer, S David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

Pierre Tassel, Martin Gebser, and Konstantin Schekotihin. An end-to-end reinforcement learning approach for job-shop scheduling problems based on constraint programming. In *International Conference on Automated Planning and Scheduling*, volume 33, pp. 614–622, 2023.

Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:13–23, 2021.

Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does zero-shot reinforcement learning exist? In *International Conference on Learning Representations (ICLR)*, 2023.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 2015.

Chenguang Wang, Zhouliang Yu, Stephen McAleer, Tianshu Yu, and Yaodong Yang. ASP: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(6): 4102–4114, 2024a.

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, 2024b.

Xuan Wu, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L Maskell, You Zhou, and Di Wang. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *IEEE Transactions on Neural Networks and Learning Systems*, 2026.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):5057–5069, 2021.

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 (ICLR)*, 2026.

Yubin Xiao, Di Wang, Boyang Li, Mingzhao Wang, Xuan Wu, Changliang Zhou, and You Zhou. Distilling autoregressive models to obtain high-performance non-autoregressive solvers for vehicle routing problems with faster inference speed. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 38, pp. 20274–20283, 2024.

Hegen Xiong, Shuangyuan Shi, Danni Ren, and Jinjin Hu. A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 142:105731, 2022.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Zikang Yu, Jinbiao Chen, and Jiahai Wang. Combination-of-experts with knowledge sharing for cross-task vehicle routing problems. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 41802–41825, 2026.

Erdong Yuan, Liejun Wang, Shuli Cheng, Shiji Song, Wei Fan, and Yongming Li. Solving flexible job shop scheduling problems via deep reinforcement learning. *Expert Systems with Applications*, 245:123019, 2024.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1621–1632, 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 (ICLR)*, volume 2024, pp. 45696–45716, 2024a.

Cong Zhang, Zhiguang Cao, Yaoxin Wu, Wen Song, and Jing Sun. Learning topological representations with bidirectional graph attention network for solving job shop scheduling problem. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 4192–4208, 2024b.

Zhi Zheng, Changliang Zhou, Xialiang Tong, Mingxuan Yuan, and Zhenkun Wang. UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:6081–6125, 2024.

Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Xu Chi. MV-MoE: Multi-task vehicle routing solver with mixture-of-experts. In *International Conference on Machine Learning (ICML)*, pp. 61804–61824, 2024a.

Jianan Zhou, Yaoxin Wu, Zhiguang Cao, Wen Song, Jie Zhang, and Zhiqi Shen. Collaboration! towards robust neural methods for routing problems. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:121731–121764, 2024b.

A Implementation and Experimental Details

A.1 Model and Training Configuration

Forward map. SchedulerFM uses latent dimension $d = 64$. Each operation node carries eight features (ready and unready indicators, processing time, waiting time, remaining job work, job-ready offset, absolute remaining job work, and number of remaining predecessors), and each machine node carries four (idle and busy indicators, remaining processing time, and idle time). The forward encoder consists of three pre-LayerNorm graph-attention blocks with hidden dimension 128 and four attention heads. Within each block, an operation attends over the remaining operations of its job (itself included) and additionally receives the embedding of the machine it requires, while a machine attends over the remaining operations queued on it. Attention logits follow GAT (Veličković et al., 2018): for node embeddings $\boldsymbol{h}_i, \boldsymbol{h}_k$, head-specific vectors $\boldsymbol{a}_{\mathrm{src}}, \boldsymbol{a}_{\mathrm{dst}}$ and a shared linear map W, the logit is $\mathrm{LeakyReLU}(\boldsymbol{a}_{\mathrm{src}}^{\top} W \boldsymbol{h}_i + \boldsymbol{a}_{\mathrm{dst}}^{\top} W \boldsymbol{h}_k)$ with slope 0.2. Each block adds a two-layer feed-forward layer, and both sublayers use residual connections. Mean and max pooling over operation and machine embeddings give the global graph embedding. Each of the two forward heads is a two-layer MLP (128 hidden units) over the candidate operation's embedding, the graph embedding, and $\boldsymbol{z}$.

Backward map and baseline. The backward encoder combines the event vector, global statistics of the successor state, and a two-layer heterogeneous GIN (Xu et al., 2019) with hidden dimension 64, which uses relation-specific two-layer MLPs with sum aggregation over operations of the same job, operation-to-machine and machine-to-operation edges, a learnable self-weight $(1 + \epsilon)$, and mean pooling. Nine of the 64 coordinates of $\mathbf{B}(o)$ are fixed statistics of the successor state (partial makespan, a lower bound, total remaining work, the maximum, mean and standard deviation of remaining machine load, total idle time, completion fraction, and workload spread); the other 55 are produced by a two-layer MLP (128 hidden units, LayerNorm, $\tanh$) over the event, these statistics, and the HGIN embedding. The result is normalized onto $\mathcal{Z} = \{\boldsymbol{z} \in \mathbb{R}^d : \|\boldsymbol{z}\|_2 = \sqrt{d}\}$. Event coordinates are divided by the instance's total processing time, and all event and analytic features are standardized with statistics estimated once from the replay buffer. The baseline $b(s, \boldsymbol{z}, e)$ embeds $(s, \boldsymbol{z})$, using the forward encoder's graph embedding, and e with two-layer MLPs of width 128, and combines them with a third. The model has 970,718 parameters.

Training data. Training instances follow the generator of L2D (Zhang et al., 2020): processing times are drawn uniformly from $\{1, \ldots, 98\}$ and each job's machine order is a uniformly random permutation. We use 512 instances for each of 12 shapes (6×6, 10×5, 15×5, 20×5, 10×10, 15×10, 20×10, 30×10, 15×15, 20×15, 20×20, 30×20), 6,144 in total, which yield a replay buffer of 1.33M transitions (Appendix A.2).

Optimization. Each minibatch contains 128 transitions from a single instance shape, sampled across scheduling depths. The networks are optimized with Adam for 16,000 updates using Polyak-averaged target networks; Table 5 lists the hyperparameters. We keep the checkpoint (update 12,000 of 16,000) whose greedy makespan-anchor rollouts have the lowest mean makespan on 384 held-out generated instances; no public benchmark instance is used for training or checkpoint selection. Training takes 1,366 s on one NVIDIA RTX 5070 Ti, replay collection 145 s, and the CP-SAT demonstrations 1,207 s of CPU time.

A.2 Replay Buffer Configuration

Behavior policies. Each generated training instance contributes one complete episode. The behavior policy of episode i is the $(i \bmod 16)$-th entry of a fixed cycle consisting of a uniform-random dispatcher and the 15 priority dispatching rules of Table 6, so each policy generates 32 of the 512 episodes of every shape. This gives 6,144 episodes (384 of them random) and 1,298,432 transitions. Every rule dispatches the eligible operation with the smallest priority key, with ties broken by operation index.

CP-SAT demonstrations. The first 16 training instances of each shape (192 in total) are additionally solved for makespan by CP-SAT (one interval per operation, job precedences, and one no-overlap constraint per machine) with a 20 s limit, 8 workers, and seed 0; 139 are solved to proven optimality. CP-SAT schedules are in general not non-delay, so each is projected into the non-delay action space

Table 5: **Training hyperparameters.**

Optimizer	Adam
Learning rate ($\mathbf{F}$, b / $\mathbf{B}$)	10^{-4} / $\frac{1}{3} \times 10^{-4}$
Minibatch	128 transitions of one instance shape
Updates / checkpoint interval	16,000 / 2,000
Polyak rate of target networks	0.005
Gradient-norm clip	10
Boltzmann temperature τ	0.2
Discount γ	1
λ_{ortho} / λ_Q / λ_b	1 / 1 / 10^{-3}
Samples for standardizers and $\mathbf{\Lambda}$	4,096 replay transitions
Random seed	42

Table 6: **Priority dispatching rules of the replay buffer.** Each rule dispatches the eligible operation with the smallest key. p is the operation's processing time, p^+ that of its job successor (0 for the last operation), W and R the processing time and number of operations remaining in its job (itself included), t^{rdy} its job's ready time, and L the unscheduled processing time queued on its machine.

Rule	Key	Rule	Key
SPT	p	LPT	$-p$
LWKR	W	MWKR	$-W$
LOR	R	MOR	$-R$
FIFO	t^{rdy}	LIFO	$-t^{\text{rdy}}$
LTWR	L	MTWR	$-L$
SNPT	p^+	LNPT	$-p^+$
SPNEXT	$p + p^+$	LPNEXT	$-(p + p^+)$
MWKRPO	$-W/R$		

by 32 guided rollouts that dispatch the eligible operation the solver starts earliest (ties by operation index). The first rollout follows this order exactly, while the others replace the guided action with a uniformly random eligible action at 10% of the decisions. The best projection is added only if its makespan is strictly below that of every behavior policy of the cycle on the same instance; 150 of the 192 pass (35,413 transitions), for 1,333,845 transitions in total.

A.3 Forward–Backward Training Details

We give the target construction and the auxiliary terms omitted from Section 4.3. Throughout, an overbar denotes a Polyak-averaged target network, $\delta_i \in \{0, 1\}$ indicates that transition i is terminal, and N_{b} is the minibatch size.

Bootstrapped target. At the successor state s', the target policy is the latent-conditioned Boltzmann policy as

$$\widetilde{\pi}_{\boldsymbol{z}}(a \mid s') \propto \exp\left(\frac{\min_{u\in\{1,2\}} \bar{\mathbf{F}}^{(u)}_{\boldsymbol{z}}(s', a)^\top \boldsymbol{z}}{\tau\sqrt{d}}\right), \quad a \in \mathcal{A}(s'), \tag{14}$$

with $\tau = 0.2$, and the target in Equation (11) is

$$\bar{m}_{ij} = (1 - \delta_i) \min_{u\in\{1,2\}} \Big[\Big(\sum_{a'\in\mathcal{A}(s'_i)} \widetilde{\pi}_{\boldsymbol{z}_i}(a' \mid s'_i)\, \bar{\mathbf{F}}^{(u)}_{\boldsymbol{z}_i}(s'_i, a')\Big)^\top \bar{\mathbf{B}}(o_j) + \bar{b}(s'_i, \boldsymbol{z}_i, e_j)\Big]. \tag{15}$$

The learned measure is therefore that of the softened policy $\widetilde{\pi}_{\boldsymbol{z}}$; at test time the policy acts greedily (Equation (8)).

Auxiliary terms. Let $\mathbf{B} \in \mathbb{R}^{N_{\text{b}}\times d}$ stack the backward representations $\mathbf{B}(o_j)$ of a minibatch and $\mathbf{C} = \mathbf{N}_{\text{b}}^{-1}\mathbf{B}^\top\mathbf{B}$. First, the orthonormality penalty is

$$\mathcal{L}_{\text{ortho}} := \|\mathbf{C} - \mathbf{I}\|_F^2. \tag{16}$$

Second, we impose one-step TD consistency on the scalar value of the reward represented by $\boldsymbol{z}$,

$$r_{\boldsymbol{z}}(o) = \boldsymbol{z}^\top \mathbf{C}^+ \mathbf{B}(o), \tag{17}$$

where $\mathbf{C}^+$ is the Moore–Penrose pseudo-inverse, so that $\mathbb{E}_\rho[r_{\boldsymbol{z}}\mathbf{B}] = \mathbf{C}\mathbf{C}^+\boldsymbol{z}$, which equals $\boldsymbol{z}$ on the span of the backward representations. Treating $r_{\boldsymbol{z}}$ as a constant, let $Q_i^{(u)} = \mathbf{F}_{\boldsymbol{z}_i}^{(u)}(s_i, a_i)^\top \boldsymbol{z}_i + \mathbf{N}_{\mathrm{b}}^{-1} \sum_j b(s_i, \boldsymbol{z}_i, e_j)\, r_{\boldsymbol{z}_i}(o_j)$; then

$$\mathcal{L}_Q = \frac{1}{d} \sum_{u=1}^{2} \mathbb{E}_i \Big[\big(Q_i^{(u)} - r_{\boldsymbol{z}_i}(o_i) - (1-\delta_i) \min_{u'} \bar{Q}_i^{(u')} \big)^2 \Big], \tag{18}$$

where $\bar{Q}_i^{(u')}$ has the same form, evaluated at s_i' with the target networks and the Boltzmann-expected forward head. Finally, the baseline penalty is $\mathcal{L}_b := \mathbb{E}_{i,j}\big[b(s_i, \boldsymbol{z}_i, e_j)^2\big]$. The complete training loss can be summarized as follows:

$$\mathcal{L} := \mathcal{L}_{\mathrm{FB}} + \lambda_{\mathrm{ortho}} \mathcal{L}_{\mathrm{ortho}} + \lambda_Q \mathcal{L}_Q + \lambda_b \mathcal{L}_b, \tag{19}$$

with $\lambda_{\mathrm{ortho}} = 1$, $\lambda_Q = 1$ and $\lambda_b = 10^{-3}$.

A.4 Latent Search Configuration

Initial population. The four reward-derived anchors use the one-hot weights $\boldsymbol{w} = \boldsymbol{e}_1, \ldots, \boldsymbol{e}_4$ in Equation (12). Since $\mathbf{F}$ reads the normalized latent and the greedy policy is invariant to a positive rescaling of $\boldsymbol{z}$, projecting $\boldsymbol{\Lambda w}$ onto $\mathcal{Z}$ does not change the induced policy. The initial population consists of these four anchors followed by latents drawn uniformly on $\mathcal{Z}$ (normalized Gaussian samples), each evaluated by one complete greedy rollout; ties among candidate operations are broken by operation index.

CMA-ES. After the initial population is evaluated, CMA-ES is initialized at the mean of its eight best candidates, projected onto $\mathcal{Z}$, with unit step size and identity covariance. Each subsequent generation samples 16 points in $\mathbb{R}^d$, projects them onto $\mathcal{Z}$, and ranks them by the realized cost of their rollouts. We use the CMA-ES implementation of the `cmaes` library without learning-rate adaptation, with a seed derived from the instance and the run seed.

Budgets. The standard budget is $32 + 30 \times 16 = 512$ rollouts per instance, in a single search without restarts. The 3×512 budget pools three independent searches with different seeds and keeps the best schedule. On the extreme-scale instances (Table 2), the budget is 512 for 100×100, $64 = 32 + 2 \times 16$ for 100×1000, and $32 = 16 + 1 \times 16$ for 1000×100, whose initial population holds the four anchors and 12 uniform latents.

Encoder window. On the extreme-scale instances only, the forward encoder sees the first 20 remaining operations of each job, the longest job length seen in pretraining. Dispatch candidates always lie inside this window, and the environment is unchanged. No other benchmark has jobs longer than 20 operations, so the window has no effect there.

A.5 Applying SchedulerFM to FJSSP

In FJSSP, operation $O_{j,k}$ can run on any machine i in an eligible set $\mathcal{M}_{j,k} \subseteq \mathcal{M}$, with processing time $p_{j,k,i}$. No FJSSP training or parameter update is performed: the frozen policy $\pi_{\boldsymbol{z}}$ selects which job to dispatch, and a parameterized machine-assignment rule selects the machine.

Decision process. At each decision step, the assignment rule first chooses a machine for the next operation of every unfinished job. The decision time is the earliest start among these choices, and the candidate actions are the jobs whose chosen machine lets them start at that time. The policy selects one candidate, whose operation is dispatched on the chosen machine. Because the rule may choose a machine that is still busy when the job becomes ready, the resulting schedules are not restricted to non-delay schedules.

Assignment rule. The rule ranks the eligible machines of job j's next operation by

$$\kappa(i) = \beta_1(S_i + p_i) + \beta_2 S_i + \beta_3 p_i + \beta_4 L_i + \beta_5 \mathbb{I}\{S_i > t_j^{\mathrm{rdy}}\}, \tag{20}$$

Table 7: **FJSSP machine-assignment rules.** Weights $\boldsymbol{\beta}$ of the named rules used to initialize the joint search, in the order of Equation (20).

Rule	β_1 (finish)	β_2 (start)	β_3 (processing)	β_4 (load)	β_5 (busy)
ECT	1	0	0	10^{-6}	0
Fastest	0	10^{-6}	1	0	0
Free-SPT	0	1	1	10^{-6}	10^{3}
Free-LL	0	1	10^{-6}	1	10^{3}
ECT-load	1	0	0	0.5	0
ECT-patient	1	-0.5	0	0	0
ECT-impatient	1	0.5	0	0	0
Balanced	0.5	0	0.5	0.5	0

where S_i and p_i are the earliest start time and the processing time of the operation on machine i, t_j^{rdy} is the ready time of job j, and L_i is the unscheduled workload of machine i under the provisional assignment described below. Time quantities are divided by the longest processing time of the instance. The machine with the smallest key is selected, with ties broken by machine index. Table 7 lists eight named members of this family.

Graph input. The encoder reads a JSSP graph in which each unscheduled operation is placed on a provisional machine, the fastest eligible one (ties to the least statically loaded, then the lowest index), with the corresponding processing time.

Joint search. The weights $\boldsymbol{\beta} \in \mathbb{R}^5$ are appended to $\boldsymbol{z}$, and CMA-ES searches the resulting 69-dimensional vector with the protocol of Appendix A.4 and the same budget of 512 rollouts. The 32 initial candidates pair each of the eight rules of Table 7 with each of the four reward-derived anchors, with no uniformly sampled candidates. Each candidate is projected onto a sphere as a whole, and its $\boldsymbol{z}$ part is then renormalized onto $\mathcal{Z}$; since the key in Equation (20) is invariant to a positive rescaling of $\boldsymbol{\beta}$, the projection does not change the rule.

A.6 Benchmark and Evaluation Details

JSSP benchmarks. The main JSSP evaluation contains 170 public instances from four benchmark families. LA contributes 40 instances across 10×5, 15×5, 10×10, 20×5, 15×10, 20×10, 15×15, and 30×10. SWV contributes 20 instances across 20×10, 20×15, and 50×10. TA contributes 30 instances across 50×15, 50×20, and 100×20. DMU contributes 80 instances across 20×15, 20×20, 30×15, 30×20, 40×15, 50×15, 40×20, and 50×20. These are exactly the instances reported in Table 1.

Extreme-scale benchmarks. We evaluate the first five instances each at 100×100, 100×1000, and 1000×100 from Da Col & Teppan (2022). The reference values are the current best-known upper bounds used in Table 2.

FJSSP benchmarks. We evaluate the 15 Brandimarte instances (mk01–mk15) and the edata, rdata, and vdata sets of Hurink et al. (1994), each containing 66 instances. We exclude `orb7`, whose zero-duration operation is treated as ineligible by the released DANIEL and ReSched parsers, leaving 65 per set and 210 instances in total. Performance is measured by makespan gap to the corresponding best-known solution. DANIEL and ReSched are trained on 15×10 instances and report the best of 100 sampled schedules (superscript and subscript in Table 3); SPT, MOR and MWR are the dispatchers released with DANIEL.

Cross-objective benchmarks. We evaluate 58 FT, LA, ABZ, and ORB instances under total flow time, total tardiness, weighted tardiness, and maximum tardiness, with makespan included as a control. Due dates are $d_j = \mathrm{round}\left(1.5 \sum_k p_{j,k}\right)$, and weights $\omega_j \in \{1, 2, 4\}$ are drawn with probabilities $(0.2, 0.6, 0.2)$ from a seed fixed per instance. The 22 dispatching rules are the random dispatcher, the 15 rules of Table 6, EDD, MDD, slack per remaining processing time, critical ratio, ATC and weighted ATC; for each instance and objective we keep the best one. COMPASS runs its own latent search from its makespan checkpoint with the objective as fitness. Because standard-

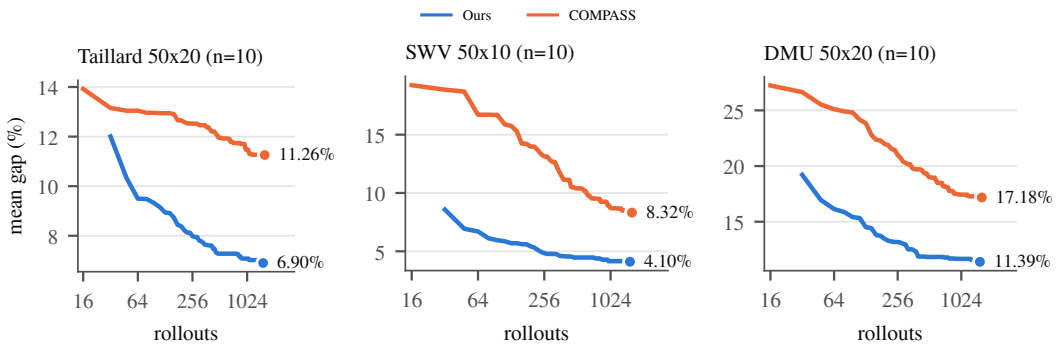


Figure 3: **Test-time compute scaling.** Mean makespan gap (%) as a function of rollout evaluations per instance on representative TA, SWV, and DMU settings. Pretrained model parameters remain fixed throughout. Lower is better.

ized best-known solutions are not available for these alternative objectives, we use the result of an unrestricted one-hour CP-SAT solve under the same objective as the reference value.

Non-delay CP-SAT. Table 1 also reports CP-SAT restricted to the same non-delay schedule class used by SchedulerFM.

B Additional Experiments

B.1 Test-Time Compute Scaling

We study whether the same frozen SchedulerFM checkpoint continues to improve as more rollout evaluations are allocated to latent-space optimization. We vary only the test-time rollout budget and compare with COMPASS under matched or comparable search budgets. Figure 3 shows that increasing the rollout budget improves SchedulerFM across the three representative settings. Thus, additional test-time computation can be converted into better schedules without updating the pretrained model.

B.2 Effect of Replay-Buffer Diversity

We examine how the diversity of scheduling behaviors in the reward-free replay buffer affects downstream performance. We train SchedulerFM with different numbers of priority dispatching rules while keeping the training instances, architecture, and optimization budget fixed, and evaluate each checkpoint on the same 170 JSSP instances as Table 1. Table 8 shows that performance is relatively stable across the tested replay mixtures, with the full $k = 16$ mixture obtaining the best overall average gap: 8.01%, compared with 8.14% for $k = 4$ and 8.16% for $k = 1$. The effect is therefore modest overall.

B.3 Comparison of Latent-Space Optimizers

We compare CMA-ES with random search, Gaussian-process Bayesian optimization (GP-BO), and the cross-entropy method (CEM) under the same budget of 512 rollout evaluations. All methods use the same pretrained SchedulerFM checkpoint and the same initial population of reward-derived anchors and uniformly sampled latent strategies. Table 9 shows that CMA-ES achieves the lowest average gap on every benchmark family and overall, reaching 8.01% compared with 9.08% for CEM, 11.24% for GP-BO, and 11.54% for random search. This shows that the benefit of test-time optimization is not explained by additional sampling alone: adaptively refining the latent search distribution uses the rollout budget more effectively.

Table 8: **Effect of replay-buffer diversity.** Average gap (%) after 512 episodes for models trained with replay from k behaviour policies, on the instances of Table 1. k=16 is used in the released pretrained model in the main text.

			Behaviour policies k		
Family	Size	n	1	4	16
LA	10×5	5	2.76	**2.60**	**2.60**
	15×5	5	0.31	**0.00**	**0.00**
	10×10	5	4.92	**4.24**	5.82
	20×5	5	0.40	**0.00**	**0.00**
	15×10	5	6.54	7.15	**6.03**
	20×10	5	7.23	6.75	**6.30**
	15×15	5	9.04	**8.04**	8.23
	30×10	5	**0.00**	**0.00**	**0.00**
	Avg	40	3.90	**3.60**	3.62
SWV	20×10	5	10.99	**10.61**	11.05
	20×15	5	13.28	12.75	**12.57**
	50×10	10	4.00	**3.96**	4.49
	Avg	20	8.07	**7.82**	8.15
TA	50×15	10	5.25	5.54	**4.53**
	50×20	10	7.70	**6.96**	7.36
	100×20	10	1.39	**0.91**	1.06
	Avg	30	4.78	4.47	**4.32**
DMU	20×15	10	11.91	11.89	**11.53**
	20×20	10	13.44	12.87	**12.84**
	30×15	10	12.98	12.59	**12.43**
	30×20	10	**14.50**	15.03	14.71
	40×15	10	**8.44**	9.16	8.76
	50×15	10	**6.59**	7.51	7.25
	40×20	10	**12.94**	13.50	13.21
	50×20	10	11.88	12.39	**11.77**
	Avg	80	11.58	11.87	**11.56**
All		170	8.16	8.14	**8.01**

B.4 Latent Strategy Visualization

Figure 4 provides a qualitative view of the learned strategy space by comparing strategies associated with dispatching rules, randomly sampled latent strategies, rule mixtures, and strategies selected by test-time optimization.

C Additional Related Work

Neural combinatorial optimization (NCO) learns neural heuristics for solving combinatorial optimization problems, either by constructing solutions directly or by improving existing ones. Early work explored supervised learning from high-quality solutions, including Pointer Networks for routing (Vinyals et al., 2015; Milan et al., 2017) and supervised approaches for scheduling (Ingimundardottir & Runarsson, 2018). Subsequent methods increasingly adopted reinforcement learning, with the Attention Model (Kool et al., 2019) and POMO (Kwon et al., 2020) establishing influential constructive paradigms for routing. This line has since been extended through improved architectures, training strategies, and search procedures for routing and related COPs (Grinsztajn et al., 2023; Chalumeau et al., 2023; Drakulic et al., 2023; Chen et al., 2023; Goh et al., 2024; Fang et al., 2024; Xiao et al., 2024; Zhou et al., 2024a;b; Bi et al., 2024; Wang et al., 2024a; Zheng et al., 2024; Chen et al., 2025b;a). A more detailed survey of NCO methods can be found in (Wu et al., 2026).

Beyond constructive policies, neural improvement methods iteratively refine candidate solutions and have been studied for both routing (Wu et al., 2021; Ma et al., 2021; 2023; Kong et al., 2024) and scheduling (Falkner et al., 2022; Zhang et al., 2024a). Other recent approaches reduce reliance on externally generated labels by using solutions produced during training itself. For example, self-labeling and self-improvement methods exploit locally improved solutions as training targets (Luo

Table 9: **Comparison of latent-search strategies.** Average gap (%) after 512 episodes for different search algorithms over z, on the instances of Table 1. All share the trained model and the initial population; GP-BO and CEM use four rounds of 128.

Family	Size	n	Random	GP-BO	CEM	CMA-ES
LA	10×5	5	3.63	3.33	2.85	**2.60**
	15×5	5	0.34	0.49	0.09	**0.00**
	10×10	5	5.70	5.05	**3.96**	5.82
	20×5	5	0.05	0.05	**0.00**	**0.00**
	15×10	5	8.56	7.40	7.40	**6.03**
	20×10	5	10.10	10.56	8.02	**6.30**
	15×15	5	10.19	10.22	8.57	**8.23**
	30×10	5	0.19	0.16	**0.00**	**0.00**
	Avg	40	4.84	4.66	3.86	**3.62**
SWV	20×10	5	15.80	15.57	12.48	**11.05**
	20×15	5	19.09	17.92	14.84	**12.57**
	50×10	10	6.41	6.29	5.10	**4.49**
	Avg	20	11.93	11.52	9.38	**8.15**
TA	50×15	10	8.46	7.87	6.29	**4.53**
	50×20	10	10.97	10.27	8.49	**7.36**
	100×20	10	3.82	2.96	1.67	**1.06**
	Avg	30	7.75	7.03	5.48	**4.32**
DMU	20×15	10	16.62	15.93	13.25	**11.53**
	20×20	10	16.61	15.65	13.85	**12.84**
	30×15	10	17.67	17.70	14.39	**12.43**
	30×20	10	18.80	19.91	16.42	**14.71**
	40×15	10	13.73	13.44	10.35	**8.76**
	50×15	10	11.11	10.70	8.12	**7.25**
	40×20	10	18.62	17.84	13.74	**13.21**
	50×20	10	16.45	17.16	13.60	**11.77**
	Avg	80	16.20	16.04	12.96	**11.56**
All		170	11.54	11.24	9.08	**8.01**

et al., 2025; Corsini et al., 2024), while BOPO (Liao et al., 2025) learns from objective-induced preferences among sampled solutions. CoEKS (Yu et al., 2026) further investigates combining complementary optimization knowledge within a unified NCO framework.

NCO has also been extensively developed for scheduling problems. Representative constructive methods include approaches for JSSP (Zhang et al., 2020; Park et al., 2021b; Jeon et al., 2023; Tassel et al., 2023; Iklassov et al., 2023; Ho et al., 2024) and FJSSP (Song et al., 2023; Yuan et al., 2024), alongside improvement-based methods such as L2S (Zhang et al., 2024a) and self-labeling approaches such as SLIM (Corsini et al., 2024).

Generalist Learners for NCO. More recently, NCO has begun to move from problem-specific solvers toward models that can be reused across heterogeneous combinatorial tasks. GOAL (Drakulić et al., 2025) develops a generalist solver based on mixed-attention blocks that accommodate different combinations of node-, edge-, and instance-level features. UCPO (Fang et al., 2026) studies a unified preference-optimization framework for adapting pretrained neural solvers across constrained combinatorial problems. These generalist approaches share the goal of increasing reuse across tasks. By contrast, SchedulerFM pursues a complementary direction: rather than primarily sharing an architecture across problem formulations, it learns a reward-free latent space of scheduling behaviors that can be adapted at test time to new instance scales, related scheduling problems, and unseen downstream objectives without updating the pretrained network.

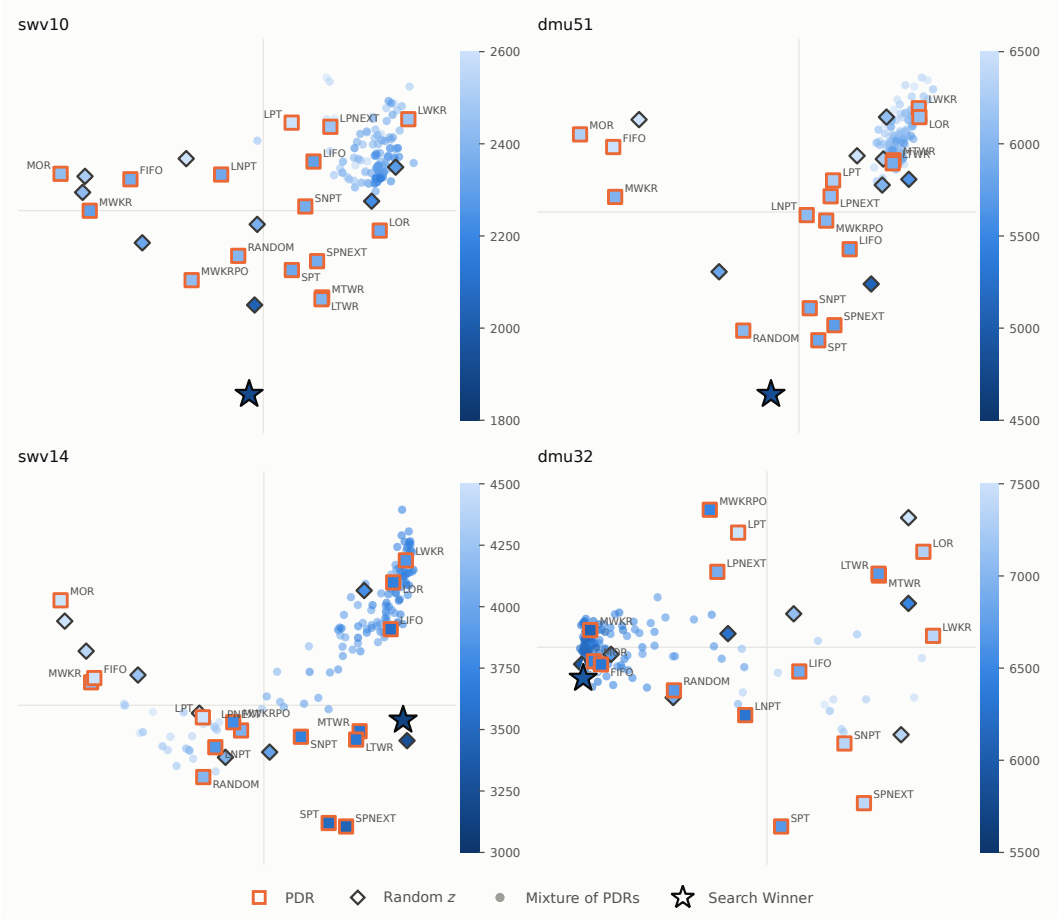


Figure 4: **Latent strategy visualization.** Episodes of the frozen model, centered by the instance-level average rule embedding and projected onto the top two principal components of the rule embeddings. Fill indicates makespan, with darker points corresponding to shorter schedules.