SAPO: SELF-ANCHORED PREFERENCE OPTIMIZATION FOR COLD-START NEURAL SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Constructive neural solvers for scheduling problems (e.g., JSSP, FJSP) face a critical *cold-start dilemma*: training from scratch lacks an informative reference prior, making standard preference optimization difficult to apply effectively. While recent reference-free methods alleviate the dependency on expert policies, they may exhibit two stage-specific weaknesses: *premature convergence* associated with deterministic winner selection in early training, and *policy drift* when no explicit trust region is available during late-stage refinement. To bridge this gap, we propose **S**elf-**A**nchored **P**reference **O**ptimization (**SAPO**), a stage-adaptive framework that dynamically constructs the missing prior from the training trajectory itself. SAPO introduces a dual-phase mechanism: (1) Distribution-Smoothed Exploration, which samples winners from a decaying top-K pool to broaden early winner supervision; and (2) Self-Anchored Refinement, which utilizes the historical-best policy as a moving anchor to provide an anchor-relative signal during refinement. Across six JSSP benchmark families, SAPO yields lower mean gaps than BOPO under a matched fixed-checkpoint decoding audit and shows improved stability on the recorded late-stage trajectory, while controlled FJSP/TSP studies provide preliminary evidence of transfer to related constructive tasks without systematic inference overhead.

1 INTRODUCTION

Job-shop scheduling (JSSP) is a central NP-hard problem characterized by tight precedence and machine-disjunctive constraints. While exact solvers offer precision, they scale poorly; conversely, hand-crafted heuristics often lack generalization across varying instance sizes and distributions. Neural Combinatorial Optimization (NCO) has emerged as a promising data-driven alternative, learning constructive policies via deep reinforcement learning Kool et al. (2018); Kwon et al. (2020). However, training these policies from scratch remains challenging. The final objective (e.g., makespan) is only revealed upon completing a solution, resulting in delayed feedback and weak credit assignment. Furthermore, raw objective magnitudes vary significantly across problem scales, complicating reward calibration. This difficulty extends to flexible job-shop scheduling (FJSP) and routing problems such as the traveling salesman problem (TSP), whose policies likewise construct complete schedules or tours before the final task objective is available. Nevertheless, once two complete solutions are obtained, their objectives induce an unambiguous within-instance ordering that is invariant to positive rescaling.

Preference optimization addresses these issues by leveraging within-instance ordinal information (e.g., $y_w \succ y_l$), rather than calibrated absolute returns Rafailov et al. (2023); Ethayarajh et al. (2024). This approach avoids hand-designed reward shaping and relies on within-instance rankings rather than calibrated cross-instance objective magnitudes. While Direct Preference Optimization (DPO) is attractive for its ability to optimize policies without an explicit reward model, it relies on a reference policy (π_{ref}) to regularize the learning process Li et al. (2025). In Large Language Model (LLM) alignment, the reference policy is naturally provided by a pre-trained model. However, in constructive NCO, we face a *cold-start regime*: training begins without a pre-trained or expert policy to serve as an informative prior. Using an untrained policy as a reference provides no structural guidance, making standard DPO difficult to apply effectively.

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.

Recent reference-free objectives, such as BOPO Liao et al. (2025), attempt to bypass this dependency. BOPO's "best anchor" is a solution-level winner used to construct pairs, not a policy-level reference. However, we focus on two potential stage-specific weaknesses of these methods when applied to cold-start scheduling: (1) Early-stage Premature Concentration: Deterministically selecting the single "best" candidate from a stochastic, untrained policy can create a noisy supervision signal and narrow the winner distribution too quickly, risking premature convergence to incidental local optima. (2) Late-stage Policy Drift: As the policy improves, the absence of a policy-level reference means the optimization lacks an explicit trust-region signal. The loss constrains relative probabilities but does not explicitly anchor the policy to the best validated behavior and may permit forgetting or drift during late-stage refinement. Figure 1 provides descriptive evidence for the late-stage pattern.

To bridge this gap, we propose **Self-Anchored Preference Optimization (SAPO)**, a stage-adaptive framework that constructs the missing prior dynamically from the training trajectory itself. SAPO operates in two phases: Distribution-Smoothed Exploration (phase 1) and Self-Anchored Refinement (phase 2). In phase 1, we sample winners from a decaying top-K candidate pool to broaden the early preference-pair distribution before a reliable policy emerges. In phase 2, once useful behavior is established, we utilize the historical-best policy (validated on a hold-out set) as a moving self-anchor. This provides a dynamic reference signal intended to discourage drift during refinement without modifying the inference architecture.

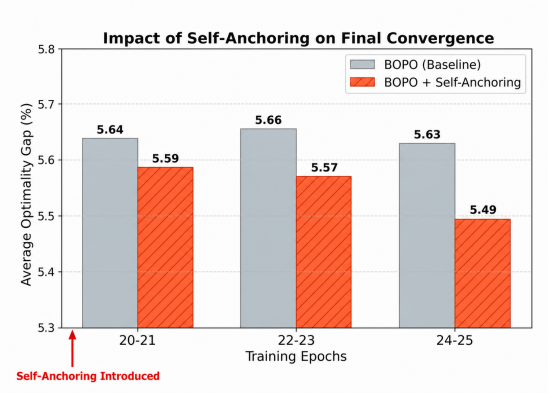


Figure 1: Late-stage validation gaps on the recorded JSSP trajectory. Bars show non-overlapping 2-epoch averages over epochs 20–25; self-anchoring begins at epoch 20, and lower is better.

Our contributions are summarized as follows:

- We characterize the cold-start dilemma in preference optimization: the tension between mitigating premature target concentration in early training and restoring an informative reference signal for late-stage stability.
- We introduce SAPO, which combines top-K distribution smoothing with a historical-best policy anchor to satisfy these requirements without modifying the inference procedure.
- We evaluate SAPO on six JSSP benchmark families with per-instance results, matched decoding-repeat analysis, and paired-instance uncertainty estimates. Trajectory and timing diagnostics probe refinement behavior and cost, while auxiliary FJSP/TSP checks assess the boundary of transfer beyond JSSP.

2 Related Work

Scheduling and Neural Combinatorial Optimization. Classical JSSP methods include dispatching rules such as SPT, exact branch-and-bound or constraint-programming solvers, and metaheuristics such as genetic algorithms, tabu search, and simulated annealing Pinedo & Hadavi (1992); Panwalkar & Iskander (1977); Carlier (1982); Brucker et al. (1994); Van Laarhoven et al. (1992); Nowicki & Smutnicki (1996). Exact methods can become computationally costly at scale, whereas hand-crafted heuristics often require instance-specific tuning. NCO instead learns constructive or improvement policies from data. Attention-based solvers and POMO established routing-style constructive learning Kool et al. (2018); Kwon et al. (2020). For JSSP, L2D and ScheduleNet learn graph-based dispatching policies Zhang et al. (2020); Park et al. (2021); L2S and TBGAT learn local improvement policies over complete schedules Zhang et al. (2024a;b); and SLIM uses self-labeling to train a constructive solver Corsini et al. (2024). These methods primarily change the architecture, search operator, or supervision. SAPO retains the MGL scheduler and decoder used by BOPO, its closest baseline, and changes training-time preference construction and regularization.

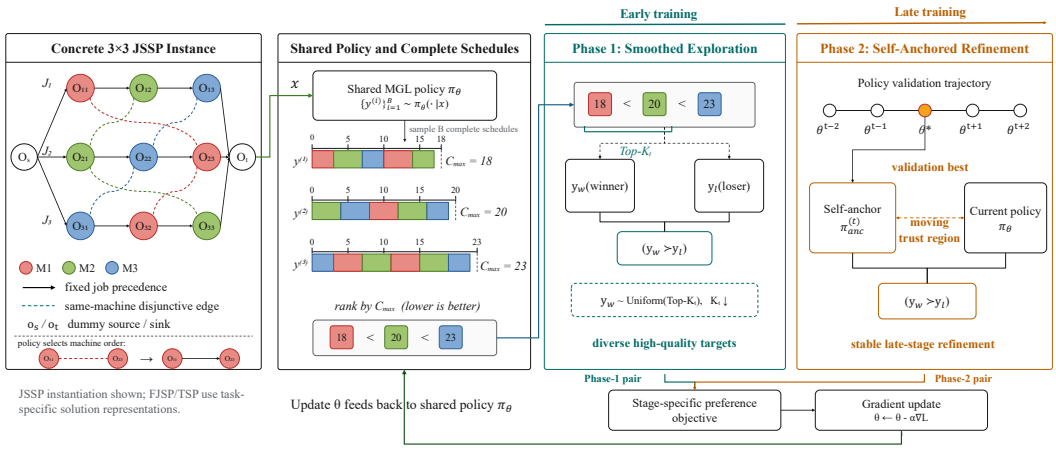


Figure 2: SAPO framework. Phase 1 samples winners from a decaying top-K_t pool for distribution-smoothed exploration; Phase 2 uses a validation-selected historical-best policy as a moving self-anchor for refinement. Candidate generation, policy architecture, and inference remain unchanged.

Preference Optimization for CO/NCO. DPO optimizes preference pairs through a reference-constrained objective Rafailov et al. (2023), while PIPA clarifies the role of the reference policy as a prior over feasible generations Li et al. (2025). Reference-free LLM objectives such as SimPO and SimPER remove the explicit reference term, but are typically applied by fine-tuning pretrained models and therefore inherit an informative initialization Meng et al. (2024); Xiao et al. (2025). Online preference optimization instead updates the policy using preference data generated by the evolving policy. IPO-MD connects online IPO with Nash mirror descent Calandriello et al. (2024), while mirror-descent-based preference learning has also been extended to multi-turn trajectories Shani et al. (2024).

In CO, BOPO uses best-anchored pair construction and objective-guided scaling for JSP, TSP, and FJSP Liao et al. (2025). Its anchor is a sampled solution used to construct preference pairs, rather than a policy distribution in the log-ratio objective. PO4COPs formulates entropy-regularized preference learning and integrates local search into fine-tuning to generate higher-quality pairs for TSP, CVRP, and FFSP Pan et al. (2025). Related work also includes POCCO for conditional multi-objective CO Fan et al. (2025) and VAGPO for vision-augmented group preferences in routing Liu et al. (2025). Unlike online alignment methods, SAPO obtains preference labels directly from the combinatorial objective without human annotations or a learned preference model. Unlike existing CO methods, it converts validation history into a policy-level reference for cold-start refinement while leaving candidate generation and the deployed solver unchanged.

3 Preliminaries and Problem Formulation

3.1 Job Shop Scheduling Problem (JSSP)

We consider a standard $N \times M$ JSSP instance x with jobs $\mathcal{J} = \{J_1, \ldots, J_N\}$ and machines $\mathcal{M} = \{M_1, \ldots, M_M\}$. Each job J_i comprises an ordered sequence of operations $O_i = (o_{i,1}, \ldots, o_{i,M})$; operation $o_{i,k}$ must run non-preemptively on machine $\mu_{i,k}$ for duration $p_{i,k}$.

The problem is represented by a disjunctive graph $\mathcal{G} = (\mathcal{V}, \mathcal{C} \cup \mathcal{D})$ Zhang et al. (2020). Nodes $\mathcal{V}$ are operations, directed conjunctive arcs $\mathcal{C}$ encode job precedence, and undirected disjunctive edges $\mathcal{D}$ connect operations competing for the same machine.

The objective is to assign start times $S_{i,k}$ that minimize the makespan $C_{\max} = \max_{i,k}(S_{i,k} + p_{i,k})$, subject to two constraint families: (1) *Job precedence*: $S_{i,k+1} \geq S_{i,k} + p_{i,k}$; and (2) *Machine non-overlap*: for operations $o_{i,k}$ and $o_{j,l}$ assigned to the same machine, either $S_{i,k} + p_{i,k} \leq S_{j,l}$ or $S_{j,l} + p_{j,l} \leq S_{i,k}$.

A feasible machine ordering orients every edge in $\mathcal{D}$ without creating a directed cycle; minimizing the longest resulting path is equivalent to minimizing $C_{\max}$. In a constructive policy setting, the agent must sequentially select the next operation to schedule from a set of eligible candidates. Since the global objective $C_{\max}$ is only revealed after the full sequence is generated, the agent faces a sparse reward problem with high variance. This motivates the shift from value-based learning to preference-based learning, which relies on relative comparisons rather than absolute cost estimation.

3.2 Preference-based Reinforcement Learning

To mitigate the difficulty of reward calibration across different instance scales, Preference-based RL (PbRL) learns from ordinal feedback rather than absolute returns. Instead of predicting the makespan directly, the agent learns to distinguish between a preferred solution y_w (winner) and a dispreferred solution y_l (loser) for the same instance x, denoted as $y_w \succ y_l$.

We model the preference probability using the Bradley-Terry (BT) model. Assuming a latent utility function $r^*(x, y)$ (e.g., the negative makespan), the probability that y_w is preferred over y_l is given by:

$$P(y_w \succ y_l|x) = \sigma(r^*(x, y_w) - r^*(x, y_l)) \tag{1}$$

where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic sigmoid function. Positive affine transformations of the cost preserve the induced winner–loser ordering, although the numerical Bradley–Terry probability depends on the scale chosen for the latent utility. Preference learning therefore avoids directly comparing raw makespan magnitudes across different JSSP sizes by relying on within-instance ordinal labels.

3.3 DPO as Prior-Informed Inference

DPO Rafailov et al. (2023) streamlines this process by optimizing the policy directly, without an explicit reward-modeling stage. Under the PIPA view Li et al. (2025), it admits a prior-informed probabilistic-estimation interpretation. For a fixed instance x, coefficient $\beta > 0$, and a reference satisfying $\pi_{\text{ref}}(y|x) > 0$ on the support of π, consider

$$\max_{\pi} \ \mathbb{E}_{y\sim\pi(\cdot|x)}[r^*(x, y)] - \beta D_{\text{KL}}(\pi(\cdot|x) \,\|\, \pi_{\text{ref}}(\cdot|x)) \,. \tag{2}$$

Its optimizer follows the Gibbs distribution $\pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x) \exp\left(\frac{r^*(x,y)}{\beta}\right)$, where $Z(x)$ normalizes the distribution. Here, π_{ref} defines the support and relative prior mass of candidate solutions, while the exponential term incorporates utility evidence. Replacing the optimizer by the trainable policy π_θ, and substituting into Eq. (1) cancels the instance-dependent $Z(x)$ and yields

$$\begin{aligned} \mathcal{L}_{\text{DPO}}(\theta) &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}_p}\big[\log \sigma(\beta\Delta_{\text{DPO}})\big], \\ \Delta_{\text{DPO}} &= \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}. \end{aligned} \tag{3}$$

Under this interpretation, the reference policy π_{ref} acts as a regularizer or a local trust-region analogue, discouraging the optimized policy from drifting too far from the reference distribution or forgetting previously acquired behavior. If we set π_{ref} to a uniform distribution (or remove it), the DPO objective reduces to a reference-free margin loss similar to BOPO. Such a loss lacks an explicit mechanism for anchoring the policy to the historical-best behavior during late-stage refinement and may therefore permit the instability discussed in the Introduction. SAPO aims to address this cold-start gap by dynamically constructing a meaningful self-anchor.

4 Methodology

Building on the prior-informed interpretation above, SAPO separates two roles that are easily conflated: Phase 1 changes the distribution of preference pairs, whereas Phase 2 changes the objective by introducing a policy-level reference constructed from the training trajectory.

As illustrated in Figure 2, both phases generate the same candidate batch for each instance and, following BOPO, sample the loser uniformly from all candidates with a strictly worse objective

value than the selected winner. For $t < T_{\text{switch}}$, Phase 1 samples a winner from a high-quality top-K_t set while retaining the reference-free loss. For $t \geq T_{\text{switch}}$, Phase 2 restores the best-in-batch winner and evaluates DPO-style log-ratios relative to a frozen validation-best checkpoint. The checkpoint changes only after a validation improvement, yielding a piecewise-constant reference for refinement.

4.1 Reference-Constraint Motivation

Let x be an instance, y a complete solution, and $c \in \{0, 1\}$ indicate whether y appears on the preferred or rejected side under the induced pair-construction process. Denote the corresponding class-conditional solution distributions by $p^+(y|x) = P(y|x, c = 1)$ and $p^-(y|x) = P(y|x, c = 0)$.

4.1.1 Preference as Density Ratio Estimation.

To connect the pairwise Bradley–Terry model to the PIPA class-conditional view, consider the Bayes-optimal logistic score $s(x, y)$ with $P(c = 1|x, y) = \sigma(s(x, y))$. Its logit is

$$\begin{aligned} s(x, y) &= \log \frac{P(c = 1|x, y)}{P(c = 0|x, y)} \\ &= \log \frac{P^+(y|x)}{P^-(y|x)} + \log \frac{P(c = 1|x)}{P(c = 0|x)}. \end{aligned} \tag{4}$$

The class-prior log-odds depend on x but not on y, so they cancel when two solutions for the same instance are compared; no balanced-class assumption is required. We may therefore use the centered density-ratio score

$$\widetilde{r}(x, y) = \log \frac{P^+(y|x)}{P^-(y|x)}. \tag{5}$$

Using this centered score in the Bradley–Terry model makes the pair probability depend on $\widetilde{r}(x, y_w) - \widetilde{r}(x, y_l)$. This identity neither makes reference-free training ill-posed by itself nor establishes convergence; it only exposes the role of the denominator distribution.

4.1.2 From a Reference-Free Margin to an Informative Reference.

Without coupling $P^-(y|x)$ to a prior, a pairwise margin may increase even when $\pi_\theta(y_w|x)$ falls, provided $\pi_\theta(y_l|x)$ falls faster; it therefore need not keep the policy near a validated incumbent. This motivates introducing an informative denominator after the policy has learned a useful solution distribution.

DPO: Anchoring via Reference Model. Within the PIPA parameterization, identifying $P^+(y|x)$ with $\pi_\theta(y|x)$ and constraining $P^-(y|x) = \pi_{\text{ref}}(y|x)$ recovers the anchor-relative score and DPO loss in Eq. 3. The reference therefore ties all comparisons to a fixed policy distribution rather than only to the two sampled trajectories.

BOPO as a Reference-free Limit. For a fixed-length finite solution space $\mathcal{Y}(x)$, a constant reference $\pi_{\text{ref}}(y|x) = |\mathcal{Y}(x)|^{-1}$ cancels from the pairwise log-ratios. BOPO uses this reference-free structure with an objective-guided pair scale:

$$\begin{aligned} \Delta_{\text{BOPO}} &= \log \pi_\theta(y_w|x) - \log \pi_\theta(y_l|x), \\ \mathcal{L}_{\text{BOPO}}(\theta) &= -\mathbb{E}_{\mathcal{D}_p}[\log \sigma(\lambda_{w,l} \Delta_{\text{BOPO}})]. \end{aligned} \tag{6}$$

Here $\lambda_{w,l} > 0$ denotes BOPO's objective-guided pair scale.

Observed Late-stage Pattern. Without an explicit reference distribution, the loss contains no anchor-relative term tying the current policy to the best validated checkpoint. The recorded JSSP trajectory shows that this can coincide with late-stage oscillation or stagnation, motivating the moving-anchor design below; this is an empirical failure mode, not a universal claim about reference-free objectives.

SAPO retains this reference-free form while constructing a useful policy checkpoint, then introduces that checkpoint as a reference in Phase 2.

4.2 PHASE 1: DISTRIBUTION-SMOOTHED EXPLORATION

In early training, an informative policy reference is unavailable. Phase 1 therefore retains the reference-free loss but changes the preference-pair distribution to broaden early winner supervision.

For each instance, let $C_x(y)$ denote the minimized task objective, let $\mathcal{B}_t(x)$ be the sampled candidates ranked by objective value, and let $\mathcal{S}_t(x)$ contain its top K_t entries. Independently for each x, we sample one winner uniformly and define its strictly worse candidate set as follows:

$$\begin{aligned} \mathcal{S}_t(x) &= \{y \in \mathcal{B}_t(x) : \mathrm{rank}_x(y) \le K_t\}, \\ \mathcal{L}_t(x, y_w) &= \{y \in \mathcal{B}_t(x) : C_x(y) > C_x(y_w)\}, \\ q_t(y_w|x, \mathcal{B}_t) &= \begin{cases} |\mathcal{S}_t(x)|^{-1}, & y_w \in \mathcal{S}_t(x), \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{7}$$

The loser is then drawn as $y_l \sim \mathrm{Uniform}(\mathcal{L}_t(x, y_w))$, so every candidate with a strictly worse objective than y_w has equal selection probability. Here K_t decays linearly from K_{start} to K_{end} before T_{switch}. Thus Phase 1 optimizes Eq. 6 under a time-varying pair distribution induced by q_t, rather than changing the reference term itself.

Winner-selection Entropy. The greedy rule has zero winner-selection entropy. Conditional on a fixed pool with $|\mathcal{S}_t(x)| = K_t$, the proposed rule has $H_{q_t}(y_w|x, \mathcal{S}_t) = \log K_t$. This is the entropy of preference construction, not of the full policy π_θ. The draw reuses the existing candidate batch and requires no additional decoder calls; deployment inference is unchanged.

4.3 PHASE 2: SELF-ANCHORED REFINEMENT

At $t \ge T_{\text{switch}}$, SAPO restores the best-in-batch winner $y_w = \arg\min_{y \in \mathcal{B}_t(x)} C_{\max}(y)$ and replaces the constant-reference interpretation with a validation-selected policy snapshot.

Let $V(\theta_s)$ be the validation gap recorded at checkpoint s, and let $\mathcal{H}_t$ contain the checkpoints evaluated before update t. The historical best checkpoint available to that update and its frozen anchor are

$$\theta_t^* \in \arg\min_{s \in \mathcal{H}_t} V(\theta_s), \qquad \pi_{\text{anc}}^{(t)} = \mathrm{stopgrad}(\pi_{\theta_t^*}). \tag{8}$$

The anchor is initialized from the best pre-switch checkpoint and held fixed during each gradient update. At a scheduled validation probe, a checkpoint enters $\mathcal{H}_t$ only if it is evaluated, and it becomes the next anchor only when its gap is lower than the historical best. For the resulting Phase-2 pair distribution $\mathcal{D}_t$, the SAPO loss is

$$\begin{aligned} \Delta_{\text{anc}}^{(t)} &= \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{anc}}^{(t)}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{anc}}^{(t)}(y_l|x)}, \\ \mathcal{L}_{\text{SAPO}}(\theta) &= -\mathbb{E}_{\mathcal{D}_t}[\log \sigma(\beta \Delta_{\text{anc}}^{(t)})]. \end{aligned} \tag{9}$$

Moving-anchor Interpretation. Under the KL-regularized derivation, Eq. equation 9 is a plug-in objective with a piecewise-constant moving reference. Its pairwise log-ratios provide an anchor-relative signal analogous to a local trust region; SAPO neither executes an explicit KL-constrained step nor guarantees global convergence. Because the anchor changes only at improving validation probes, it does not follow transient mini-batch fluctuations. SAPO changes only training-time pair construction and regularization; the decoder, sampling budget, and inference procedure remain unchanged.

Full pseudocode for SAPO, including candidate generation, stage-dependent pair construction, loss evaluation, parameter updates, and validation-gated anchor maintenance, is provided in Appendix B, Algorithm 1.

5 EXPERIMENTS

We use JSSP as the primary benchmark and organize the evaluation around three questions: (i) how SAPO compares with established solvers under zero-shot benchmark transfer; (ii) whether its two

stages improve refinement behavior without changing deployment cost; and (iii) whether the same training principle remains useful in related scheduling and routing domains. We therefore evaluate SAPO primarily on JSSP, with FJSP and TSP serving as controlled transfer checks under matched BOPO/SAPO budgets within each domain.

5.1 Experimental Setup

5.1.1 Implementation Details

SAPO is implemented in PyTorch with the same MGL (Multi-layer GAT-LSTM) policy architecture used by the BOPO baseline Corsini et al. (2024); Liao et al. (2025). For the controlled BOPO–SAPO comparison, we keep the training data, encoder, decoder, candidate budget, and inference procedure fixed; the preference-pair construction and Phase-2 reference are the intended differences. Main JSSP training uses 25 epochs, 16 comparison samples, Adam with learning rate 2×10^{-5}, and $T_{\text{switch}} = 20$. Experiments run on an Intel Xeon Platinum 8358P CPU and a single NVIDIA RTX 3090 GPU; remaining hyperparameters are in the supplement.

5.1.2 Datasets

Following SLIM Corsini et al. (2024), training uses 30,000 synthetic JSSP instances uniformly split across 10×10, 15×10, 15×15, 20×10, 20×15, and 20×20, with 100 validation instances per size. Testing is zero-shot: the model is trained only on this synthetic set and evaluated without fine-tuning on Taillard (TA) Taillard (1993), DMU Demirkol et al. (1998), ABZ Adams et al. (1988), Lawrence (LA) Lawrence (1984), SWV Storer et al. (1992), and YN Yamada & Nakano (1992).

5.1.3 Baselines

We compare against the dispatching rule SPT, constructive neural solvers (L2D Zhang et al. (2020), SchN Park et al. (2021), SLIM Corsini et al. (2024), and BOPO Liao et al. (2025)), general-purpose exact/constraint solvers under their reported limits (Gurobi and OR-Tools), and RL-based improvement methods (L2S Zhang et al. (2024a) and TGA Zhang et al. (2024b)). Table 1 uses the reported 500- and 5,000-step L2S settings and the $B' = 512$ SLIM sampling setting; unavailable benchmark–method pairs are shown as dashes. BOPO is the closest training baseline because it also optimizes neural scheduling preferences and is evaluated under the same architecture and decoding budgets as SAPO. The remaining published baselines provide broader context under their source settings rather than controlled objective-only comparisons.

5.1.4 Metrics and Statistical Protocol

We report average optimality gap, $\frac{1}{N}\sum_i (C_i - C_i^*)/C_i^* \times 100\%$, and the unweighted macro mean over six families; lower is better. BOPO/SAPO checkpoints are validation-selected, while other baselines retain source settings. The supplementary fixed-checkpoint audit repeats $B' = 512$ sampling over seeds $\{10, 20, 30, 40, 50\}$ and reports family means $\pm$ sample standard deviations, quantifying decoding rather than training-run variability; greedy decoding is deterministic. Transfer checks use three matched training runs; latency uses ten paired repetitions and 100,000-resample bootstrap intervals.

5.2 Experimental Results

Main Results. Table 1 gives the primary per-size JSSP evaluation; BOPO–SAPO conclusions use the supplementary matched audit because published baselines retain source settings. Across five matched $B' = 512$ seeds, SAPO lowers all six family means and the macro gap (8.061 versus 8.373, -0.312 percentage points). These repeats quantify decoding, not training-run, variability; full statistics appear in the supplement.

Problem		Exact Solver		RL-based Improvement			Greedy Constructive						Sampling Constructive		
Set	Size	Gurobi	OR-T	$L2S_{500}$	TGA_{500}	$L2S_{5k}$	SPT	L2D	SchN	SLIM	BOPO	SAPO	SLIM	BOPO	SAPO
TA	15×15	0.1	0.1	9.3	8.0	6.2	25.8	26.0	15.3	13.8	15.2	14.2	6.5	**6.4**	6.7
	20×15	3.2	0.2	11.6	9.9	8.3	32.9	30.0	19.4	15.0	15.6	14.6	8.8	7.8	**7.4**
	20×20	2.9	0.7	12.4	10.0	9.0	27.8	31.6	17.2	15.2	15.7	14.9	9.0	8.5	**8.4**
	30×15	10.7	2.1	14.7	13.3	9.0	35.1	33.0	19.1	17.1	17.4	14.8	10.6	9.8	**9.5**
	30×20	13.2	2.8	17.5	16.4	12.6	34.4	33.6	23.7	18.5	17.9	18.5	12.7	12.8	**12.2**
	50×15	12.2	3.0	11.0	9.6	4.6	24.1	22.4	13.9	10.1	10.4	9.3	4.9	5.4	**4.4**
	50×20	13.6	2.8	13.0	11.9	6.5	25.6	26.5	13.5	11.6	12.2	10.8	7.6	7.5	**6.8**
	100×20	11.0	3.9	7.9	6.4	3.0	14.4	13.6	6.7	5.8	6.1	5.1	2.1	2.2	**1.2**
	Avg.	8.4	2.0	12.2	10.7	7.4	27.5	27.1	16.1	13.4	13.8	12.8	7.8	7.6	**7.0**
DMU	20×15	5.3	1.8	20.9	21.6	14.2	28.0	39.0	-	18.0	17.6	16.6	11.3	11.5	**10.7**
	20×20	4.7	1.9	19.5	22.0	13.9	31.3	37.7	-	19.4	22.8	20.5	12.3	13.2	**11.9**
	30×15	14.2	2.5	26.4	26.4	20.2	31.5	42.0	-	21.8	21.8	21.5	14.0	**13.9**	14.0
	30×20	16.7	4.4	27.0	27.7	20.5	34.4	39.7	-	25.7	24.4	22.3	15.8	16.7	**15.3**
	40×15	16.3	4.1	26.2	26.1	19.9	24.0	35.6	-	17.5	17.8	16.0	10.9	11.1	**10.7**
	40×20	22.5	4.6	29.4	29.9	22.5	37.2	39.6	-	22.2	26.0	22.1	**14.8**	16.1	15.2
	50×15	14.9	3.8	26.2	26.2	22.1	24.8	36.5	-	15.7	17.3	14.9	10.6	11.0	**10.3**
	50×20	22.5	4.8	32.2	32.3	24.3	30.1	39.5	-	22.4	24.9	21.5	**15.0**	16.2	15.5
	Avg.	14.6	3.5	25.9	26.5	19.7	30.2	38.7	-	20.3	21.6	19.4	13.1	13.7	**12.9**
LA	10×5	0.0	0.0	2.1	2.1	1.8	14.8	14.3	12.1	9.3	7.3	6.3	**1.1**	1.9	1.9
	15×5	0.0	0.0	0.0	0.0	0.0	14.9	7.8	2.7	2.6	0.5	0.1	**0.0**	**0.0**	**0.0**
	20×5	0.0	0.0	0.0	0.0	0.0	13.7	6.3	3.6	2.1	0.0	0.6	**0.0**	**0.0**	**0.0**
	10×10	0.0	0.0	4.4	1.8	0.9	15.7	23.7	11.9	8.9	8.1	5.8	2.5	2.5	**1.9**
	15×10	0.0	0.0	6.4	3.6	3.4	28.7	27.2	14.6	11.6	11.6	10.2	5.0	4.8	**4.4**
	20×10	0.0	0.0	7.0	5.0	2.6	33.4	24.6	15.7	12.1	12.2	11.7	5.6	4.8	**4.6**
	30×10	0.0	0.0	0.2	0.0	0.0	13.9	8.4	3.1	2.0	3.2	1.4	**0.0**	**0.0**	**0.0**
	15×15	0.0	0.0	7.3	5.5	5.9	24.6	27.1	16.1	13.6	13.0	12.4	5.6	5.6	**4.5**
	Avg.	0.0	0.0	3.4	2.3	1.8	20.0	17.4	10.0	7.8	7.0	6.1	2.5	2.4	**2.1**
ABZ	10×10	-	0.0	2.8	-	1.4	13.0	14.8	6.1	6.4	10.0	7.8	-	**1.8**	2.4
	20×15	-	1.0	11.9	-	8.6	33.3	24.9	20.5	15.0	17.1	17.9	-	9.5	**8.8**
	Avg.	-	0.6	8.3	-	5.7	25.2	22.1	14.7	11.6	14.2	13.9	-	6.5	**6.2**
SWV	20×10	-	0.1	29.6	23.0	17.8	26.3	41.4	34.4	23.2	20.1	20.8	12.0	**11.1**	12.1
	20×15	-	2.5	25.5	23.7	17.0	32.0	40.6	30.5	27.1	25.0	21.0	**13.5**	14.5	14.2
	50×10	-	1.6	21.4	20.3	17.1	17.0	30.8	25.3	10.6	11.8	11.4	8.0	8.0	**6.9**
	Avg.	-	1.4	24.5	21.8	17.3	23.0	35.9	28.9	17.9	17.2	16.2	10.4	10.4	**10.1**
YN	20×20	-	0.5	12.4	-	8.7	30.6	22.1	18.4	16.1	15.5	14.9	-	10.5	**9.0**
	Avg.	-	0.5	12.4	-	8.7	30.6	22.1	18.4	16.1	15.5	14.9	-	10.5	**9.0**

Table 1: Detailed JSSP results (average gap, %; lower is better). Best sampling values are bold. L2S subscripts denote steps; sampling SLIM uses $B' = 512$. BOPO/SAPO checkpoints are selected by validation performance; other baselines use source settings. "–" indicates unavailable results.

Late-stage Trajectory Diagnostic. Using the unsmoothed epoch-level validation gaps, SAPO has lower temporal gap variability over epochs 10–24 (0.0765 versus 0.1323). Over the final five epochs (21–25), SAPO also has a lower mean gap (5.46 versus 5.64). The paired epoch-wise Wilcoxon diagnostic over epochs 10–24 gives $p < 0.01$, but adjacent checkpoints are correlated; these values describe one matched trajectory and are not independent-run significance evidence. Figure 1 instead displays non-overlapping 2-epoch averages over epochs 20–25.

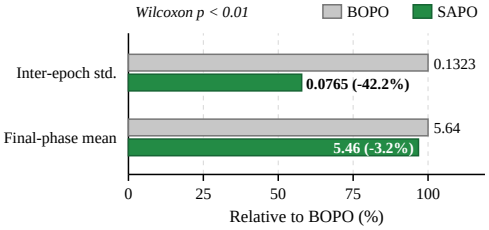


Figure 3: Late-stage validation stability. Variability and Wilcoxon diagnostics use unsmoothed epochs 10–24; the final mean uses epochs 21–25. Lower is better; correlated checkpoints make the comparison descriptive.

Inference Efficiency. Table 2 reports post-warm-up GPU latency for matched decoding on the shared 351,232-parameter network. Across nine settings, SAPO's geometric-mean overhead is $+0.24\%$, and all paired-bootstrap 95% intervals contain zero. The supplement gives full intervals and offline costs ($+5.5\%$ projected, $+6.1\%$ end-to-end, and $+0.4\%$ Phase-2 peak memory).

5.2.1 Auxiliary Transfer Checks

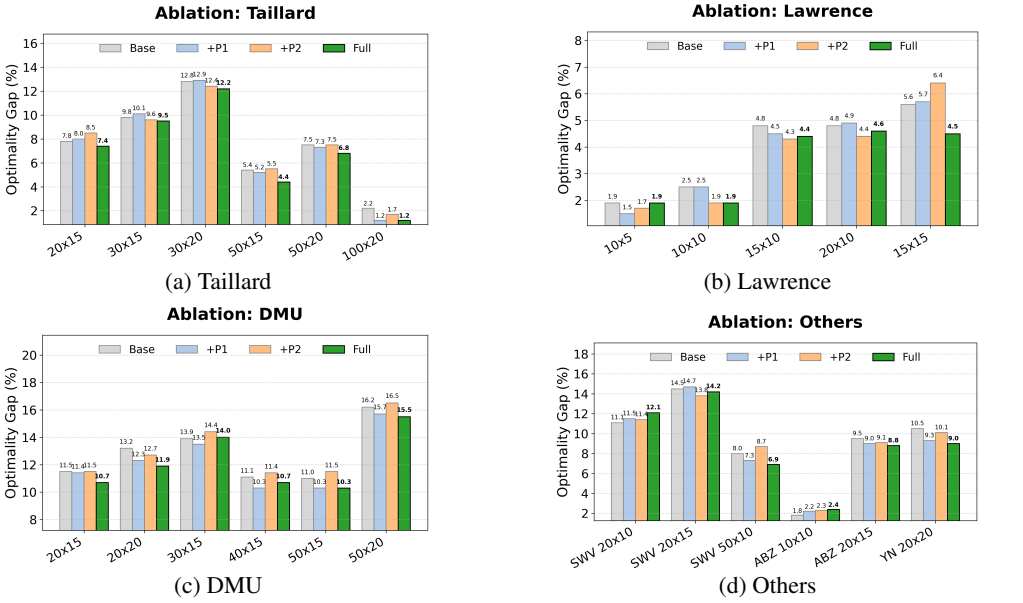
Table 3 reports matched transfer checks over three trained seeds; full settings are in the supplement. SAPO improves all FJSP and TSP-50 results; on saturated TSP-20, BOPO is slightly better without augmentation and ties with $8\times$ augmentation.

Table 2: Inference latency (seconds per instance; mean ± standard deviation over ten paired repetitions).

Set	B	BOPO	SAPO
DMU (80)	128	1.028±.036	1.034±.033
	256	1.030±.023	1.033±.023
	512	1.150±.028	1.158±.037
LA (40)	128	.236±.006	.238±.008
	256	.286±.014	.282±.008
	512	.292±.005	.293±.007
TA (80)	128	1.213±.046	1.218±.049
	256	1.240±.034	1.245±.033
	512	1.389±.046	1.389±.050

Table 3: Auxiliary FJSP/TSP gaps (%; mean ± standard deviation over three seeds; lower is better).

Setting	BOPO	SAPO
FJSP LA-e	6.211 ± 0.127	**5.922** ± 0.236
FJSP LA-r	3.828 ± 0.129	**3.678** ± 0.096
FJSP LA-v	0.539 ± 0.020	**0.526** ± 0.026
TSP-50 no aug.	0.1518 ± 0.0112	**0.1383** ± 0.0141
TSP-50 8× aug.	0.0089 ± 0.0014	**0.0053** ± 0.0003
TSP-20 no aug.	**0.0194** ± 0.0038	0.0234 ± 0.0047
TSP-20 8× aug.	0.0001 ± 0.0001	**0.0000** ± 0.0001



(a) Taillard (b) Lawrence

(c) DMU (d) Others

Figure 4: Ablation gaps for BOPO, Phase-1-only (+P1), Phase-2-only (+P2), and full SAPO; lower is better.

5.2.2 Ablation Study

Figure 4 compares full SAPO with variants using only Phase 1 (+P1) or Phase 2 (+P2). Neither stage dominates in isolation: Phase 1 often helps but is not uniformly best, while Phase 2 alone can degrade difficult cases such as DMU 30×15 (13.9% to 14.4%). Combining both stages improves the group summaries, including 13.7% to 12.9% on DMU and 10.5% to 9.0% on YN. These patterns support the complementarity of distribution-smoothed exploration and self-anchored refinement.

6 Conclusion

We introduced SAPO for cold-start preference optimization in neural scheduling without a pre-trained reference policy. SAPO derives guidance from its own training trajectory: decaying top-K winner sampling diversifies early supervision, while a validation-selected historical-best policy guides late refinement. The architecture, decoder, and inference procedure remain unchanged. Across six JSSP families, SAPO attains lower family-level mean gaps and a lower macro-average gap under the fixed-checkpoint decoding audit, together with improved stability on the recorded late-stage trajectory. Controlled FJSP/TSP checks provide preliminary evidence of transfer to related constructive tasks, and paired timing shows no systematic inference overhead. Overall, trajectory-derived self-anchoring offers a practical cold-start training principle.

AI USE STATEMENT

Generative AI tools were used solely to aid or polish the writing of this manuscript. The authors independently verified all AI-assisted content, and take full responsibility for the content of this submission.

REFERENCES

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

Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, 49(1-3):107–127, 1994.

Daniele Calandriello, Zhaohan Daniel Guo, Remi Munos, Mark Rowland, Yunhao Tang, Bernardo Avila Pires, Pierre Harvey Richemond, Charline Le Lan, Michal Valko, Tianqi Liu, Rishabh Joshi, Zeyu Zheng, and Bilal Piot. Human alignment of large language models through online preference optimisation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5409–5435. PMLR, 2024.

Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47, 1982.

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

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

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

Mingfeng Fan, Jianan Zhou, Yifeng Zhang, Yaoxin Wu, Jinbiao Chen, and Guillaume Adrien Sartoretti. Preference-driven multi-objective combinatorial optimization with conditional computation. *arXiv preprint arXiv:2506.08898*, 2025.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

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*, 33:21188–21198, 2020.

S Lawrence. An experimental investigation of heuristic scheduling techniques. *Supplement to resource constrained project scheduling*, 1984.

Junbo Li, Zhangyang Wang, and Qiang Liu. Pipa: Preference alignment as prior-informed statistical estimation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 36042–36056. PMLR, 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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 37456–37475. PMLR, 2025.

Shiyan Liu, Bohan Tan, and Yan Jin. Vagpo: Vision-augmented asymmetric group preference optimization for the routing problems. *arXiv preprint arXiv:2508.01774*, 2025.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. *Advances in Neural Information Processing Systems*, 37:124198–124235, 2024.

Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management science*, 42(6):797–813, 1996.

Mingjun Pan, Guanquan Lin, You-Wei Luo, Bin Zhu, Zhien Dai, Lijun Sun, and Chun Yuan. Preference optimization for combinatorial optimization problems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47672–47696. PMLR, 2025.

Shrikant S Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25 (1):45–61, 1977.

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

Michael Pinedo and Khosrow Hadavi. Scheduling: theory, algorithms and systems development. In *Operations research proceedings 1991: Papers of the 20th annual meeting/vorträge der 20. Jahrestagung*, pp. 35–42. Springer, 1992.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Lior Shani, Aviv Rosenberg, Asaf Cassel, Oran Lang, Daniele Calandriello, Avital Zipori, Hila Noga, Orgad Keller, Bilal Piot, Idan Szpektor, Avinatan Hassidim, Yossi Matias, and Remi Munos. Multi-turn reinforcement learning from preference human feedback. *arXiv preprint arXiv:2405.14655*, 2024.

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.

Peter JM Van Laarhoven, Emile HL Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

Teng Xiao, Yige Yuan, Zhengyu Chen, Mingxiao Li, Shangsong Liang, Zhaochun Ren, and Vasant G Honavar. Simper: A minimalist approach to preference alignment without hyperparameters. *arXiv preprint arXiv:2502.00883*, 2025.

Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *PPSN*, volume 2, pp. 281–290, 1992.

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*, 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*, 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 *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*, volume 244 of *Proceedings of Machine Learning Research*, pp. 4192–4208. PMLR, 2024b.

A SUPPLEMENTARY SCOPE

This supplement reports evidence and implementation details that are not shown in the main paper. To avoid duplication, it does not reproduce the complete JSSP benchmark table, the main-paper ablation visualization, the FJSP/TSP summary table, or the stability and latency values already reported in the main paper. It provides the expanded training algorithm, uncertainty analyses for the primary JSSP evaluation, a controlled sensitivity study of anchor-update rules, policy-entropy diagnostics, confidence intervals for inference latency, offline training-cost measurements, additional derivation steps, and reproducibility details.

B EXPANDED SAPO TRAINING ALGORITHM

Algorithm 1 gives the complete end-to-end training protocol: initialization, candidate generation and ranking, stage-dependent pair construction, per-instance loss evaluation, parameter updates, validation, and anchor maintenance. Following BOPO (Liao et al., 2025), both stages reuse the same candidate batch and sample the loser uniformly from all candidates whose objective is strictly worse than that of the selected winner. Phase 1 changes the winner distribution, whereas Phase 2 restores the best-in-batch winner and changes the preference objective through an anchor-relative log ratio. The validation-selected checkpoint is updated only at the prespecified validation probes and becomes the frozen anchor for subsequent Phase-2 updates.

Here B is the number of candidate schedules generated per instance, and $\lambda_{w,l}$ is BOPO's objective-guided scale for the selected pair.

C JSSP CHECKPOINT ROBUSTNESS

Table 4 reports a fixed-checkpoint robustness audit across all JSSP families. The checkpoints are selected according to validation performance. Unlike the primary evaluation in the main-paper benchmark table, this audit repeats $B' = 512$ sampling with matched decoding seeds $\{10, 20, 30, 40, 50\}$ and reports seed-averaged family statistics. The two summaries may therefore differ slightly because of decoding randomness. Greedy decoding is deterministic, and the sampling repeats characterize decoding variability for fixed checkpoints rather than independent training runs.

At the level of five-seed family means, SAPO is lower than BOPO on all six JSSP families. The unweighted six-family macro average is 8.061% for SAPO versus 8.373% for BOPO, a reduction of 0.312 percentage points. Because these repeats vary only decoding seeds for fixed checkpoints, they characterize decoding variability rather than independent training-run uncertainty.

Matched decoding-seed robustness

Family	BOPO	SAPO	Δ
TA	7.527 ± 0.064	7.227 ± 0.037	-0.300
DMU	13.693 ± 0.086	13.196 ± 0.076	-0.497
LA	2.356 ± 0.046	2.137 ± 0.104	-0.219
ABZ	6.533 ± 0.359	6.277 ± 0.232	-0.256
SWV	10.224 ± 0.132	10.140 ± 0.086	-0.084
YN	9.902 ± 0.271	9.386 ± 0.528	-0.516
Macro	8.373	8.061	-0.312

Table 4: **JSSP checkpoint robustness for $B' = 512$.** Family rows report mean gap (%) $\pm$ sample standard deviation over five matched decoding seeds. Macro is the unweighted average of the six displayed family means. Δ is SAPO$-$BOPO based on the reported means, so negative values favor SAPO.

Overall, the fixed-checkpoint audit gives lower SAPO mean gaps on all six JSSP families and at the macro level; its scope remains robustness to decoding randomness rather than independent retraining.

Algorithm 1: Expanded Self-Anchored Preference Optimization (SAPO)

Require: Training set $\mathcal{D}$, validation set $\mathcal{V}$, initial policy π_θ, minimized objective C_x, epochs E, candidate budget B.
Require: $T_{\text{switch}}, K_{\text{start}}, K_{\text{end}}$, validation schedule $\mathcal{T}_{\text{val}}$, learning rate η, temperature β.
Evaluate $V(\theta; \mathcal{V})$; set $(\theta^*, V^*) \leftarrow (\theta, V(\theta; \mathcal{V}))$.
for epoch $t = 0$ to $E - 1$ **do**
  Step 1: Candidate generation and ranking.
  Draw a training mini-batch $X_t \subset \mathcal{D}$.
  for each instance $x \in X_t$ **do**
    Sample $\mathcal{B}_t(x) = \{y^{(j)}\}_{j=1}^{B} \sim \pi_\theta(\cdot|x)$.
    Compute $C_x(y)$ and rank $\mathcal{B}_t(x)$ in ascending order.
  end for
  Step 2: Stage-dependent preference pairs.
  if $t < T_{\text{switch}}$ **then**
    Phase 1: Distribution-smoothed exploration.
    $K_t \leftarrow \text{LinearDecay}(K_{\text{start}}, K_{\text{end}}, t)$.
    for each instance $x \in X_t$ **do**
      $\mathcal{S}_t(x) \leftarrow \text{TopK}_{K_t}(\mathcal{B}_t(x))$.
      Sample $y_w \sim \text{Uniform}(\mathcal{S}_t(x))$.
      $\mathcal{L}_t(x, y_w) \leftarrow \{y \in \mathcal{B}_t(x) : C_x(y) > C_x(y_w)\}$.
      Sample $y_l \sim \text{Uniform}(\mathcal{L}_t(x, y_w))$.
      $\Delta_{\text{BOPO}} \leftarrow \log \pi_\theta(y_w|x) - \log \pi_\theta(y_l|x)$.
      $\mathcal{L}_x \leftarrow -\log \sigma(\lambda_{w,l} \Delta_{\text{BOPO}})$.
    end for
  else
    Phase 2: Self-anchored refinement.
    $\pi_{\text{anc}}^{(t)} \leftarrow \text{stopgrad}(\pi_{\theta^*})$.
    for each instance $x \in X_t$ **do**
      $y_w \leftarrow \arg\min_{y \in \mathcal{B}_t(x)} C_x(y)$.
      $\mathcal{L}_t(x, y_w) \leftarrow \{y \in \mathcal{B}_t(x) : C_x(y) > C_x(y_w)\}$.
      Sample $y_l \sim \text{Uniform}(\mathcal{L}_t(x, y_w))$.
      $r_t(y_w|x) \leftarrow \log \pi_\theta(y_w|x) - \log \pi_{\text{anc}}^{(t)}(y_w|x)$.
      $r_t(y_l|x) \leftarrow \log \pi_\theta(y_l|x) - \log \pi_{\text{anc}}^{(t)}(y_l|x)$.
      $\mathcal{L}_x \leftarrow -\log \sigma(\beta[r_t(y_w|x) - r_t(y_l|x)])$.
    end for
  end if
  Step 3: Policy optimization.
  $\mathcal{L} \leftarrow |X_t|^{-1} \sum_{x \in X_t} \mathcal{L}_x$.
  $\theta \leftarrow \text{Adam}(\theta, \eta, \nabla_\theta \mathcal{L})$.
  Step 4: Validation and anchor maintenance.
  if $t \in \mathcal{T}_{\text{val}}$ **then**
    Evaluate $V(\theta; \mathcal{V})$.
    if $V(\theta; \mathcal{V}) < V^*$ **then**
      Save the checkpoint and set $(\theta^*, V^*) \leftarrow (\theta, V(\theta; \mathcal{V}))$.
    end if
  end if
end for
Ensure: Validation-selected policy π_{θ^*}.

D Anchor-Update Sensitivity

We compare four Phase-2 anchor rules: *historical best*, which refreshes the anchor only after a new global validation best; *fixed*, which retains the Phase-1 snapshot; *periodic*, which refreshes from the current policy after each epoch; and *EMA*, which updates a moving average at every training step with decay 0.999. Historical best is the prespecified SAPO rule; the other three are controlled alternatives.

D.1 Controlled Restart and Evaluation Protocol

All branches start from the same epoch-19 Phase-1 checkpoint and receive the same five-epoch Phase-2 budget (global epochs 20–24). Each branch uses 30,000 training instances, 600 validation

(a) Aggregate checkpoint performance

Rule	Val. best	Selected gap	Δ [95% CI]	Final gap
Phase-1 initial	5.2148	8.248 ± 0.127	–	–
Historical best	5.1646	$\mathbf{7.964 \pm 0.076}$	$\mathbf{-0.284}$ $[\mathbf{-0.434}, \mathbf{-0.156}]$	8.289 ± 0.134
Fixed	**5.1545**	8.275 ± 0.069	$+0.027$ $[-0.098, +0.130]$	8.536 ± 0.059
Periodic	5.2081	8.332 ± 0.072	$+0.084$ $[-0.032, +0.174]$	8.168 ± 0.070
EMA	5.2148	8.248 ± 0.127	0.000 $[0.000, 0.000]$	8.571 ± 0.106

(b) Historical-best family effects

Family	Initial	Historical best	Δ_{HB} [95% CI]
TA	7.138 ± 0.052	$\mathbf{7.042 \pm 0.061}$	-0.096 $[-0.122, -0.069]$
DMU	13.198 ± 0.062	$\mathbf{12.483 \pm 0.081}$	-0.715 $[-0.839, -0.624]$
LA	2.224 ± 0.072	$\mathbf{2.150 \pm 0.049}$	-0.074 $[-0.131, -0.009]$
ABZ	6.710 ± 0.337	6.563 ± 0.116	-0.147 $[-0.374, +0.156]$
SWV	11.018 ± 0.142	$\mathbf{9.780 \pm 0.103}$	-1.238 $[-1.408, -1.096]$
YN	$\mathbf{9.198 \pm 0.377}$	9.764 ± 0.333	$+0.566$ $[+0.058, +1.074]$
Macro	8.248 ± 0.127	$\mathbf{7.964 \pm 0.076}$	-0.284 $[-0.434, -0.156]$

(c) Alternative-rule family gaps

Family	Fixed	Periodic	EMA
TA	7.175 ± 0.046	7.135 ± 0.079	7.138 ± 0.052
DMU	13.656 ± 0.067	13.516 ± 0.108	13.198 ± 0.062
LA	2.186 ± 0.049	2.242 ± 0.044	2.224 ± 0.072
ABZ	$\mathbf{5.782 \pm 0.495}$	6.378 ± 0.136	6.710 ± 0.337
SWV	11.049 ± 0.124	11.099 ± 0.138	11.018 ± 0.142
YN	9.802 ± 0.231	9.621 ± 0.258	$\mathbf{9.198 \pm 0.377}$
Macro	8.275 ± 0.069	8.332 ± 0.072	8.248 ± 0.127

Table 5: **Sensitivity to Phase-2 anchor-update rules.** Gaps are mean (%) $\pm$ sample standard deviation over five matched decoding seeds. Panel (a) gives six-family checkpoint summaries; Δ compares the validation-selected checkpoint with the common Phase-1 initial checkpoint. Panel (b) isolates the family-level historical-best effect, and panel (c) reports the alternative rules. Negative Δ_{HB} favors historical best; lower is better.

instances, training sampling budget $B = 256$, validation budget $B' = 256$, 16 preference comparisons, Adam with learning rate 2×10^{-5}, seed 0 for Python/NumPy/PyTorch, and validation probes every 2,500 training instances. Because the original Phase-1 optimizer state was unavailable, Adam is reinitialized identically for all four branches. This is therefore a controlled Phase-2 restart, not four independent end-to-end training runs.

Testing uses the six JSSP families, 229 instances, $B' = 512$, and matched decoding seeds $\{10, 20, 30, 40, 50\}$. For every rule we evaluate the common initial checkpoint, the validation-selected checkpoint, and the epoch-24 final checkpoint. Reported standard deviations and 100,000-resample bootstrap intervals quantify decoding randomness across the five matched seeds; they do not quantify training-run uncertainty.

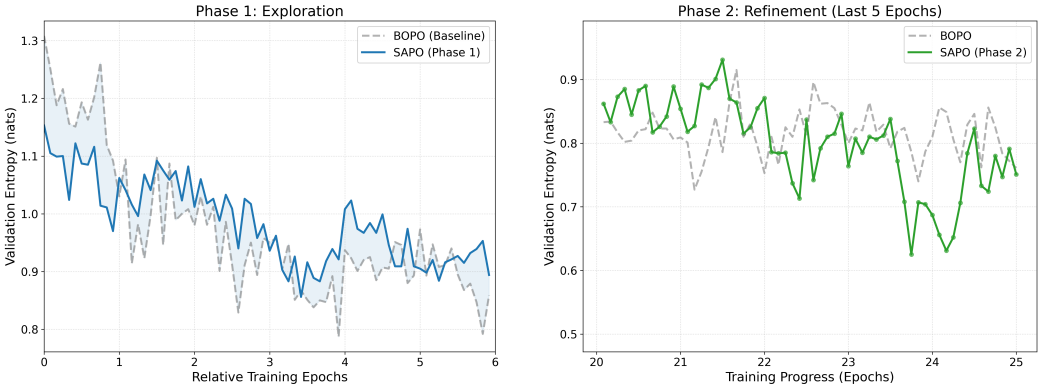
Table 5 shows that the historical-best update reduces the validation-selected macro test gap from 8.248% to 7.964%, with a matched decoding-seed difference of -0.284 percentage points (95% CI $[-0.434, -0.156]$). The corresponding intervals for fixed and periodic cross zero, while EMA never improves upon the initial validation best and therefore selects the initial checkpoint. Fixed attains the lowest validation gap, but that advantage does not transfer to the six-family test macro. Final checkpoints are included only as a robustness diagnostic: selecting periodic final because of its test value would violate the prespecified validation-only selection protocol.

At the family level, historical best improves TA, DMU, LA, and SWV, is directionally favorable but inconclusive on ABZ, and degrades YN. No update rule is uniformly best across families. The

controlled study therefore supports the prespecified historical-best rule as the most reliable aggregate choice among the evaluated alternatives while making the family-level heterogeneity explicit.

E Policy-Entropy Diagnostics

Figures 5a and 5b report validation-set empirical policy entropy along the recorded trajectories. During early training, SAPO generally maintains higher entropy than BOPO, consistent with top-K_t winner sampling exposing the policy to multiple promising candidates. This empirical policy entropy is distinct from the analytic $\log K_t$ entropy of the winner-selection rule. During late training, SAPO reaches lower entropy over portions of the self-anchored refinement trajectory while retaining temporal variation. Both figures are descriptive trajectories rather than cross-run statistics.



(a) Phase 1: distribution-smoothed exploration.

(b) Phase 2: self-anchored refinement.

Figure 5: **Policy-entropy dynamics.** Validation-set policy entropy along the recorded trajectories. SAPO generally preserves more early-stage diversity in Phase 1 and reaches lower entropy over portions of Phase 2; both comparisons are descriptive.

F Efficiency Details

F.1 Inference-Latency Confidence Intervals

The main paper reports the measured inference times. Here we provide the paired relative differences and confidence intervals. BOPO and SAPO deploy the same 351,232-parameter encoder–decoder and use identical instance order, decoder calls, sampling budgets, and paired seeds. Every represented shape is warmed up before timing, and `torch.cuda.synchronize()` is called immediately before and after each timed pass. Loading, parsing, warm-up, gap computation, printing, and file I/O are excluded. Measurements use an RTX 3090 with PyTorch 2.6.0+cu118, CUDA 11.8, and cuDNN 9.1; intervals use 100,000 paired-bootstrap resamples.

The geometric-mean relative difference is $+0.242\%$, and all nine confidence intervals contain zero. These paired measurements support comparable deployment latency and no measurable systematic inference overhead, not a speed advantage for either method.

F.2 Offline Training Cost

The self-anchor is training-only. We profile BOPO, SAPO Phase 1, and SAPO Phase 2 with $B = 256$ and 16 preference comparisons. Each mode receives 12 untimed warm-up steps followed by five repetitions of 120 synchronized optimizer steps, for 1,800 measured steps in total. Peak allocated memory is measured after optimizer and CUDA warm-up.

Set	B	Rel. diff.	95% CI
DMU	128	$+0.58\%$	$[-0.53, +1.79]\%$
DMU	256	$+0.25\%$	$[-0.57, +1.14]\%$
DMU	512	$+0.67\%$	$[-0.31, +1.78]\%$
LA	128	$+1.02\%$	$[-0.30, +2.17]\%$
LA	256	-1.48%	$[-2.94, +0.16]\%$
LA	512	$+0.36\%$	$[-1.36, +2.00]\%$
TA	128	$+0.40\%$	$[-1.03, +2.26]\%$
TA	256	$+0.37\%$	$[-0.25, +1.13]\%$
TA	512	$+0.02\%$	$[-1.85, +1.95]\%$

Table 6: **Inference-latency uncertainty.** Relative SAPO–BOPO difference over ten paired repetitions.

Metric	BOPO	SAPO P1	SAPO P2
s/step	0.6081 ± 0.0056	0.6189 ± 0.0134	0.7328 ± 0.0110
h/epoch	5.07	5.16	6.11
Peak MiB	4139.9	4139.9	4156.4
Overhead	0.0%	$+1.8\%$	$+20.5\%$

Table 7: **Controlled training-cost profile.** Mean $\pm$ sample standard deviation over five repeated 120-step blocks. Epoch time projects 30,000 optimizer steps.

Weighting the profile by 20 Phase-1 epochs and five Phase-2 epochs projects 126.69 hours for BOPO and 133.69 hours for SAPO, a 5.5% increase. The original full runs took 152.99 and 162.32 hours, respectively ($+6.10\%$), as a consistent single-run wall-clock cross-check that also includes validation, checkpointing, and host-side waiting. Phase 2 increases peak allocation by 16.5 MiB (approximately 0.4%).

G Additional Derivation Details

G.1 Stationary Solution of the KL-Regularized Objective

The main paper states the Gibbs-form optimizer of the KL-regularized preference-inference objective associated with DPO and PIPA (Rafailov et al., 2023; Li et al., 2025). Here we provide the omitted stationary calculation. For a fixed instance x, assume $\pi_{\mathrm{ref}}(y|x) > 0$ on the feasible support and impose $\sum_y \pi(y|x) = 1$. The Lagrangian is

$$\mathcal{J}(\pi, \lambda) = \sum_y \pi(y|x) r(x, y) - \beta \sum_y \pi(y|x) \log \frac{\pi(y|x)}{\pi_{\mathrm{ref}}(y|x)} + \lambda \left(\sum_y \pi(y|x) - 1 \right). \tag{10}$$

For every solution with positive probability,

$$\frac{\partial \mathcal{J}}{\partial \pi(y|x)} = r(x, y) - \beta \left(\log \frac{\pi(y|x)}{\pi_{\mathrm{ref}}(y|x)} + 1 \right) + \lambda = 0. \tag{11}$$

Rearranging and absorbing the terms independent of y into the normalization constant gives

$$\pi^*(y|x) = \frac{\pi_{\mathrm{ref}}(y|x) \exp(r(x, y)/\beta)}{\sum_{y'} \pi_{\mathrm{ref}}(y'|x) \exp(r(x, y')/\beta)}. \tag{12}$$

Taking reward differences for two solutions of the same instance cancels the partition function and yields the reference-relative log-ratio used by DPO-style preference losses.

G.2 Scope of the Moving-Anchor Interpretation

At Phase-2 update t, SAPO chooses its anchor only from checkpoints evaluated before that update and applies `stopgrad` to the selected policy. Consequently, the anchor is piecewise constant between validation improvements, and gradients affect only the current policy. Substituting this frozen historical-best policy for the reference yields a plug-in approximation to KL-regularized preference inference. It does not impose a hard KL constraint, guarantee monotonic validation improvement, or establish global optimality. The interpretation is local: anchor-relative log probabilities discourage uncontrolled deviation from the best validation-selected behavior available so far.

H Implementation and Reproducibility Details

H.1 Architecture and Data

All primary JSSP experiments use the same 351,232-parameter MGL-style graph encoder and LSTM decoder as the corresponding BOPO baseline. The encoder processes conjunctive job edges and disjunctive machine edges, and the decoder selects executable operations sequentially. Training uses 30,000 synthetic instances across six sizes, with 100 validation instances per size. Keeping candidate generation, architecture, and inference fixed isolates the training-time pair construction and Phase-2 reference.

H.2 JSSP Configuration

Parameter	**Value**
Optimizer	Adam
Learning rate	2×10^{-5}
Candidate rollouts B	256
Total epochs	25
Comparison samples K	16
Switch epoch T_{switch}	20
Winner-pool endpoints	$K_{\text{start}} = 8$, $K_{\text{end}} = 1$
Top-K_t schedule	Linear decay for $t < 20$
Pair scaling	Objective-guided makespan ratio
Anchor refresh	New historical validation best
Validation interval	2,500 training instances
Validation sampling size	256
Encoder hidden/output size	64/128
GAT heads	2
Decoder LSTM hidden size	64
Attention size	128
Dropout	0.0
Leaky ReLU slope	0.15

Table 8: **Primary JSSP configuration.**

H.3 Statistical Reporting Boundaries

The main JSSP checkpoint audit uses five matched decoding seeds and does not estimate independent training-run variability. The anchor study shares one Phase-1 trajectory and identically restarts Adam for the four Phase-2 rules. Its intervals likewise resample matched decoding seeds rather than training runs. Entropy curves are descriptive trajectories. Inference latency uses ten paired repetitions per setting, and the training-cost profile uses five repeated blocks per mode. These distinctions determine the scope of every uncertainty statement in this supplement.