RESHAPING GUIDANCE OBJECTIVE WITH LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimization problems typically have well-defined task objectives, yet the guidance objectives used to steer optimization algorithms toward high-quality solutions are often hand-designed or manually tuned and may be suboptimal. In this paper, we study *guidance objective reshaping* with large language models (LLMs), aiming to improve optimization outcomes while accounting for structural–numerical coupling and objective alignment, and reducing reliance on manual human effort. To this end, we formulate guidance objective reshaping as a structural-numerical optimization problem and propose *Joint Objective Reshaping (JOR)*. Specifically, JOR combines LLM-driven structural adaptation with zeroth-order numerical optimization over a reparameterized latent space, while aligning the reshaped objective by improving downstream performance and proximity to the high-quality solution region. This design enables structural–numerical modifications to translate more reliably into improvements under the task objective. Extensive experiments across combinatorial optimization, LLM inference acceleration, Bayesian optimization, and reinforcement learning reward shaping show that the guidance objectives discovered by JOR consistently steer algorithms or solvers toward better solutions across diverse settings. Further analysis demonstrates that the reshaped guidance objectives primarily improve the optimization landscape without substantially altering the high-quality solution region under the task objective. Our work establishes guidance objective reshaping as a promising paradigm for LLM-assisted search, learning, and optimization.

1 INTRODUCTION

Objectives are central to AI and optimization (Li et al., 2025a; Chen et al., 2025; Yao et al., 2025; Wei et al., 2026), as they specify what an algorithm is ultimately intended to achieve. In many systems, however, the objective used to guide computation differs from the *task objective* used to evaluate the resulting solution. Instead, algorithms often rely on an intermediate *guidance objective* to determine where to search, what to prioritize, or how to update their decisions. For example, in Bayesian optimization, the task objective is to identify the best black-box function value within a limited evaluation budget, whereas the guidance objective is an acquisition function, such as expected improvement (EI), that determines which candidate to evaluate next (Jones et al., 1998; Snoek et al., 2012). Designing effective guidance objectives has therefore been a long-standing challenge, as their quality can critically shape the underlying optimization process.

Existing methods can be broadly viewed from two stages of development. Traditional and learning-based approaches typically operate within a predefined function structure and optimize its numerical realization, such as component weights, or intrinsic-component parameters (Sorg et al., 2010; Zheng et al., 2018; Hu et al., 2020). These methods can effectively improve numerical calibration, but the functional form of the guidance objective is largely fixed in advance, limiting their ability to discover new shaping mechanisms. More recently, LLM-based methods have substantially expanded the structural search space by generating and iteratively revising executable guidance functions or objective components from task descriptions and downstream feedback (Yu et al., 2023; Xie et al., 2024; Ma et al., 2024; Han et al., 2024; Li et al., 2025a; Chen et al., 2025).

However, these works often treat structural modification and numerical calibration only loosely in conjunction, leaving several important challenges unresolved: 1) *Structural-numerical coupling*:

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.

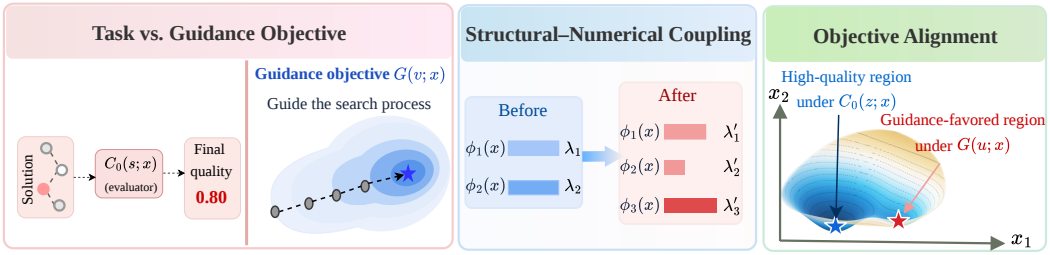

Figure 1: Illustration of guidance objective reshaping and its challenges.

structural changes may alter the interactions among active components and the optimization behavior induced by their parameters, making the previous numerical configuration unsuitable; 2) *Objective alignment*: searching solely based on downstream performance may yield reshaped objectives that favor locally optimal regions or overfit to particular optimization regimes. The solution region induced by the reshaped guidance objective should therefore consider alignment with the high-quality solution region under the task objective; and 3) *Generality*: existing approaches have largely been developed for specific problem classes or learning settings, limiting their applicability across domains. As a result, there remains a lack of a general framework for reshaping guidance objectives across diverse optimization settings while addressing the above challenges.

To address these challenges, we first formulate guidance objective reshaping as a coupled structural-numerical optimization problem and formally define its key concepts, decision variables, and optimization objective. We then introduce *Joint Objective Reshaping (JOR)*, a framework designed to address structural-numerical coupling while maintaining alignment with the high-quality solution region of the task objective. Concretely, JOR iteratively uses an LLM to propose executable modifications to the functional structure of the guidance objective, followed by zeroth-order optimization that jointly recalibrates all active numerical variables in a reparameterized latent space to account for interactions induced by each structural change. Since this recalibration does not require differentiating through the downstream algorithm, JOR is applicable to a broad range of tasks with differentiable or non-differentiable objectives and optimization procedures. Candidate updates are retained only when they improve performance under the task objective while satisfying an alignment safeguard that preserves proximity to a high-quality reference solution region within a small tolerance, thereby reducing the risk of exploiting regime-specific search dynamics and promoting robust performance across diverse conditions. Our empirical results show that the reshaped guidance objectives primarily improve the optimization landscape encountered by the algorithm while largely preserving the high-quality solution region induced by the task objective.

Contributions: 1) *Conceptually*, we investigate *guidance objective reshaping* and formulate it as a structural-numerical optimization problem, with formal definitions of its key concepts, decision variables, and optimization objective. 2) *Methodologically*, we introduce JOR, a framework that leverages LLMs to reshape guidance objectives while addressing structural-numerical coupling and objective alignment. 3) *Experimentally*, we evaluate JOR across combinatorial optimization, LLM inference acceleration, Bayesian optimization, and reinforcement learning reward shaping. JOR consistently improves downstream performance (e.g., by up to 44.7% over baselines) while also improving the underlying optimization landscape faced by the optimization algorithms.

2 Related Work

Existing work on objective reshaping spans three broad directions. Early methods focus on tuning the numerical parameters of a predefined shaping structure (Sorg et al., 2010; Zheng et al., 2018; Hu et al., 2020), whereas more recent LLM-based approaches expand the search space by generating or revising executable guidance functions, including reward programs, acquisition functions, and search heuristics (Yu et al., 2023; Xie et al., 2024; Ma et al., 2024; Aglietti et al., 2024; Lange et al., 2026; Qu et al., 2026). More recent efforts have begun to couple structural discovery with numerical adaptation, but typically depend on task-specific forms of supervision, such as trajectory preferences or visual demonstrations (Li et al., 2025a; Chen et al., 2025), which can restrict their applicability

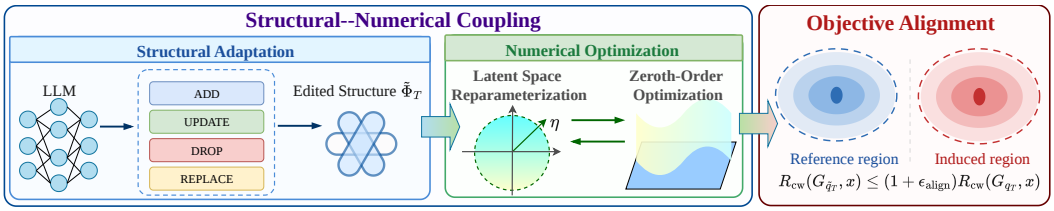

Figure 2: Overview of JOR framework.

across domains. JOR instead uses downstream performance under the original task objective as a common signal for both structural selection and numerical refinement. This shared interface allows the same reshaping framework to be instantiated across diverse guidance objectives, solvers, and downstream algorithms. A more detailed discussion is provided in Appendix A.

3 Methodology

3.1 Problem Formulation

We consider optimization problems in which the task objective is *well defined*, while the guidance objective used to steer the optimization process remains *open-ended*. Formally, let x denote a problem instance and s the final solution. The task objective is defined as $C_0(s; x)$, which remains unchanged throughout guidance objective reshaping and serves as the ultimate criterion for evaluating task performance. Let $A_{\mathcal{B}}$ denote a fixed optimization algorithm operating under a finite computational budget $\mathcal{B}$. Given a guidance objective G, the algorithm produces $s = A_{\mathcal{B}}(G; x, \omega)$, where ω captures the randomness in the optimization process. The guidance objective G therefore shapes the behavior of the optimization process in $A_{\mathcal{B}}$ and, consequently, the solution obtained.

In this paper, we study *guidance objective reshaping*. Let v denote the decision context in which the guidance objective is evaluated, such as a search candidate, acquisition point, learning transition, or decoding state. Starting from a base guidance objective G_0, we parameterize the reshaped objective by an admissible structural configuration $\Phi = \{\phi_k\}_{k=1}^{K}$ and its numerical realization $\eta = (\theta, \lambda)$. Each structural component ϕ_k has tunable parameters θ_k, while λ_k controls its contribution. The resulting guidance objective is defined as

$$G_{\Phi,\eta}(v; x) = G_0(v; x) + \sum_{k=1}^{K} \lambda_k \phi_k(v; x; \theta_k). \tag{1}$$

We formulate guidance objective reshaping as a mixed structural-numerical optimization problem. Without loss of generality, consider a problem with a minimization task objective C_0. We define the resulting structural-numerical optimization problem (with objective J) as

$$\min_{\Phi,\eta} J(\Phi, \eta) = \mathbb{E}_{x,\omega}\left[C_0\left(A_{\mathcal{B}}\left(G_{\Phi,\eta}; x, \omega\right); x\right)\right], \tag{2}$$

where J measures the expected task performance induced by the reshaped guidance objective $G_{\Phi,\eta}$ when used by the optimization algorithm $A_{\mathcal{B}}$ under a computational budget $\mathcal{B}$. To address the formulated problem, we propose JOR, which leverages LLMs to optimize the guidance objective along two complementary dimensions: structural adaptation over Φ and numerical optimization over η, as described in Section 3.2, while enforcing objective alignment as introduced in Section 3.3. Figure 2 illustrates the JOR workflow, organized around two key principles: *structural-numerical coupling* and *objective alignment*. An LLM first adapts the structure of the guidance objective, after which zeroth-order optimization refines its numerical variables η in a reparameterized latent space. Objective alignment then compares the induced solution region with the high-quality reference region to constrain alignment drift. We detail these components in the following.

3.2 Structural-Numerical Coupling

In JOR, structural adaptation modifies the functional composition of the guidance objective by adding, revising, or removing shaping components, whereas numerical optimization recalibrates the

parameters and coefficients of all active components based on downstream task performance. Importantly, a structural change can alter the relative geometry induced by the guidance objective, such as the ranking of candidate decisions, the trade-offs among competing factors, and the sensitivity to different components. In this sense, the structure determines what the numerical parameters actually mean in terms of optimization behavior. Consequently, numerical values calibrated under one structure may no longer induce the same behavior under another, motivating joint structural adaptation and numerical optimization rather than treating them as separable optimization problems. A more detailed proof is provided in Appendix B.1.

Structural Adaptation. At each iteration T, JOR uses an LLM to propose executable modifications to the current guidance objective structure Φ_T. We consider four edit operators, $\mathcal{O} = \{\mathrm{ADD}, \mathrm{UPDATE}, \mathrm{DROP}, \mathrm{REPLACE}\}$, which respectively introduce, revise, remove, or replace a shaping component. Applying a proposed edit yields a candidate structure $\widetilde{\Phi}_T$, from which the accepted update becomes Φ_{T+1}. The utility of each candidate is determined through downstream evaluation under C_0. Before numerical optimization, we apply a lightweight screening procedure to discard invalid proposals, with further details provided in Appendix B.2.

Note that a structural modification can alter the effective contribution of not only the edited component but also previously active components. Let $\eta^\star(\Phi) = \arg\min_\eta J(\Phi, \eta)$ denote the optimal numerical realization for a fixed structure Φ. After a structural update, the optimal numerical realization generally changes, such that $\eta^\star(\Phi_{T+1}) \neq \eta^\star(\Phi_T)$. Consequently, evaluating the updated structure using the previous numerical configuration may conflate the quality of the structural change with errors arising from stale numerical calibration.

Numerical Optimization. Given a screened candidate structure $\widetilde{\Phi}_T$, JOR jointly recalibrates all active numerical variables η by solving

$$\eta^\star(\widetilde{\Phi}_T) = \arg\min_\eta J(\widetilde{\Phi}_T, \eta). \tag{3}$$

The induced objective $\eta \mapsto J(\widetilde{\Phi}_T, \eta)$ is defined through the composite mapping $\eta \to G_{\widetilde{\Phi}_T, \eta} \to A_{\mathcal{B}} \to C_0$. Because $A_{\mathcal{B}}$ may involve discrete decisions, heuristic search operations, stochastic transitions, or truncated learning dynamics, $J(\widetilde{\Phi}_T, \eta)$ is generally non-differentiable and potentially non-smooth with respect to η. This motivates the use of zeroth-order optimization.

Latent Space Reparameterization. Numerical variables may differ substantially in scale and admissible range, making isotropic perturbations in the original parameter space inappropriate. We therefore optimize an *unconstrained latent vector* ξ and map it through $\mathcal{T}$ to a valid numerical configuration η under the candidate structure $\widetilde{\Phi}_T$. Let $\bar{\eta}$ denote the numerical configuration obtained after structural screening. For a fixed candidate structure, $\bar{\eta}$ remains unchanged throughout the numerical optimization and serves as the center of the latent parameterization. For variables with symmetric local refinement ranges, we define

$$\eta = \mathcal{T}(\bar{\eta}, \xi) = \bar{\eta} + b \tanh(\xi), \quad \eta, \bar{\eta}, \xi, b \in \mathbb{R}^{d_\eta}, \tag{4}$$

where d_η denotes the number of active numerical variables and b specifies their maximum allowable deviations from $\bar{\eta}$. This transformation enables isotropic perturbations in the unconstrained latent space while ensuring that every evaluated numerical configuration remains within its prescribed refinement range. We initialize numerical optimization at $\xi_0 = 0$, such that $\mathcal{T}(\bar{\eta}, \xi_0) = \bar{\eta}$. During optimization, $\bar{\eta}$ remains fixed, while the latent vector is progressively optimized. The numerical configuration at step t is therefore $\eta_t = \mathcal{T}(\bar{\eta}, \xi_t)$. For asymmetric or one-sided admissible ranges, the mapping can be modified to respect the corresponding lower and upper bounds while preserving feasibility. One such construction is provided in Appendix B.3.

Zeroth-Order Estimator. At numerical optimization step t, JOR samples perturbation directions $u_i \sim \mathcal{N}(0, I)$ and evaluates perturbations around the current latent vector ξ_t:

$$\Delta_{t,i} = J\Big(\widetilde{\Phi}_T, \mathcal{T}(\bar{\eta}, \xi_t + \sigma u_i)\Big) - J\Big(\widetilde{\Phi}_T, \mathcal{T}(\bar{\eta}, \xi_t - \sigma u_i)\Big), \tag{5}$$

where $\sigma > 0$ controls the perturbation radius in latent space. For stochastic algorithms $A_{\mathcal{B}}$, we use common random numbers (e.g., ω) for each pair of evaluations whenever applicable, so that their difference primarily reflects the effect of the perturbation rather than unrelated stochastic noise.

Moreover, since the algorithm $A_{\mathcal{B}}$ may occasionally produce large pairwise performance differences, our zeroth-order estimator clips the observed difference as $\bar{\Delta}_{t,i} = \mathrm{clip}(\Delta_{t,i}, -\tau, \tau)$ and estimates the smoothed descent direction by

$$g_\tau(\xi_t) = \frac{1}{2m\sigma} \sum_{i=1}^{m} \bar{\Delta}_{t,i} u_i, \tag{6}$$

where $\tau > 0$ is the clipping threshold, and m is the number of sampled perturbation directions used to estimate the zeroth-order descent direction. JOR then updates the latent vector through Adam (Kingma & Ba, 2014) as $\xi_{t+1} = \mathrm{Adam}\big(\xi_t, g_\tau(\xi_t)\big)$, and obtains the numerical configuration for the next optimization step through $\eta_{t+1} = \mathcal{T}(\bar{\eta}, \xi_{t+1})$. Thus, numerical optimization iteratively refines $\xi_0, \xi_1, \ldots, \xi_{t_{\max}}$ through zeroth-order optimization under a fixed parameterization centered at $\bar{\eta}$, progressively improving the numerical realization of the candidate structure $\widetilde{\Phi}_T$. Further derivations and implementation details are provided in Appendix B.3.

3.3 Objective Alignment

The structural-numerical optimization described above relies primarily on feedback from the task objective C_0. While this provides a valid optimization signal, it may be insufficient for guidance objective reshaping, as it can favor a local optimal region or overfit the reshaped objective to a particular optimization regime (e.g., a fixed computational budget $\mathcal{B}$). A reshaped guidance objective may exploit regime-specific search dynamics and achieve strong performance under the current setting without necessarily inducing an optimization landscape that consistently favors high-quality solutions. To address this challenge, JOR additionally aligns the reshaped guidance objective with a high-quality (or near-optimal) solution region defined by C_0. This alignment encourages the reshaped objective to improve not only downstream performance under the current regime, but also the underlying optimization landscape in a way that remains effective across varying conditions.

Concretely, for a guidance objective G and problem instance x, let $\mathcal{S}_G(x) = \{A_{\mathcal{B}}(G; x, \omega_i)\}_{i=1}^{P}$ denote the solution set obtained from repeated finite-budget runs. We compare this induced solution region with a high-quality reference region $\mathcal{S}_{C_0}(x)$, constructed by a task-specific procedure (see Appendix D.4) operating solely under the task objective C_0. To quantify how closely the solution region induced by G aligns with this reference region, we introduce the *cross-to-within ratio* $R_{\mathrm{cw}}(G; x) = \frac{D_{\mathrm{cross}}(G;x)}{D_{\mathrm{within}}(G;x)}$, which measures their cross-region separation relative to their intrinsic within-region variation. Specifically, we assign uniform weights to the distinct solutions in each region and define the average within-region distance D_{within} over all solution pairs, including self-pairs with zero distance, as

$$D_{\mathrm{within}}(G; x) = \frac{1}{2} \left(\frac{1}{|\mathcal{S}_{C_0}(x)|^2} \sum_{s,s' \in \mathcal{S}_{C_0}(x)} d(s, s') + \frac{1}{|\mathcal{S}_G(x)|^2} \sum_{s,s' \in \mathcal{S}_G(x)} d(s, s') \right). \tag{7}$$

The average cross-region distance D_{cross} is defined as

$$D_{\mathrm{cross}}(G; x) = \frac{1}{|\mathcal{S}_{C_0}(x)||\mathcal{S}_G(x)|} \sum_{s^{C_0} \in \mathcal{S}_{C_0}(x)} \sum_{s^G \in \mathcal{S}_G(x)} d(s^{C_0}, s^G). \tag{8}$$

When $R_{\mathrm{cw}}(G; x)$ is close to one, the average separation between the induced and reference regions is comparable to their average within-region variability, indicating stronger empirical region-level alignment. The construction of the reference region, the domain-specific distance function $d(\cdot, \cdot)$, and further details of the alignment metric are provided in Appendix D.4.

After structural adaptation and numerical optimization, let q_T denote the current accepted configuration, and $\widetilde{q}_T = (\widetilde{\Phi}_T, \eta_{t_{\max}})$ denote the candidate configuration obtained after $t_{\max}$ numerical optimization steps for the screened structure $\widetilde{\Phi}_T$ at outer iteration T. A candidate is retained only if it improves downstream task performance while preserving alignment within a small tolerance:

$$q_{T+1} = \begin{cases} \widetilde{q}_T, & \text{if } J(\widetilde{q}_T) < J(q_T),\ R_{\mathrm{cw}}(G_{\widetilde{q}_T}, x) \le (1 + \epsilon_{\mathrm{align}}) R_{\mathrm{cw}}(G_{q_T}, x), \\ q_T, & \text{otherwise}, \end{cases} \tag{9}$$

Table 1: CVRP results across scales and budgets. Gap (% ↓) is mean ± std. over three runs against LKH-3 references or best-known solutions. Time is average single-core seconds per instance.

Solver	Iteration	Method	VRP200		VRP500		VRP1000		CVRPLib	
			Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
LKH-3	100	Vanilla	1.96 ± 0.03	0.79	2.83 ± 0.04	2.02	6.56 ± 0.05	**13.0**	3.53 ± 0.05	5.61
		Claude-goal	2.05 ± 0.03	0.76	2.90 ± 0.03	1.95	6.68 ± 0.09	13.5	3.52 ± 0.03	5.49
		ShinkaEvolve	1.93 ± 0.12	0.72	2.59 ± 0.06	1.90	6.67 ± 0.14	13.2	3.38 ± 0.06	5.30
		CORAL	2.09 ± 0.06	0.72	2.89 ± 0.09	1.96	6.65 ± 0.16	13.7	3.64 ± 0.14	**5.19**
		JOR-CORAL	1.89 ± 0.02	0.65	2.53 ± 0.04	1.79	6.68 ± 0.04	13.3	$\mathbf{3.30 \pm 0.02}$	5.40
		JOR	$\mathbf{1.87 \pm 0.05}$	**0.58**	$\mathbf{2.43 \pm 0.06}$	**1.67**	$\mathbf{6.57 \pm 0.13}$	13.1	3.35 ± 0.05	5.33
	1000	Vanilla	0.57 ± 0.02	7.57	0.87 ± 0.02	18.4	1.38 ± 0.05	68.1	1.85 ± 0.04	37.6
		Claude-goal	0.66 ± 0.05	7.45	0.90 ± 0.02	17.7	1.32 ± 0.04	64.2	1.83 ± 0.02	35.8
		ShinkaEvolve	0.58 ± 0.03	7.30	0.74 ± 0.00	17.1	1.11 ± 0.02	62.1	1.75 ± 0.02	33.9
		CORAL	0.67 ± 0.09	8.32	0.91 ± 0.00	16.8	1.41 ± 0.03	63.3	1.87 ± 0.03	37.6
		JOR-CORAL	0.54 ± 0.02	6.46	0.70 ± 0.04	15.2	$\mathbf{1.06 \pm 0.01}$	59.7	1.76 ± 0.01	33.5
		JOR	$\mathbf{0.53 \pm 0.06}$	**5.88**	$\mathbf{0.65 \pm 0.02}$	**13.7**	1.09 ± 0.03	**55.0**	$\mathbf{1.72 \pm 0.05}$	**25.7**
	10000	Vanilla	0.00 ± 0.03	98.5	0.00 ± 0.02	240	0.00 ± 0.02	859	1.16 ± 0.04	429
		Claude-goal	0.03 ± 0.04	97.1	0.05 ± 0.02	236	-0.05 ± 0.01	843	1.17 ± 0.02	409
		ShinkaEvolve	0.01 ± 0.02	92.6	-0.07 ± 0.02	224	-0.18 ± 0.02	829	1.10 ± 0.01	380
		CORAL	0.06 ± 0.05	95.8	0.02 ± 0.01	222	0.00 ± 0.01	804	1.15 ± 0.02	523
		JOR-CORAL	$\mathbf{-0.01 \pm 0.02}$	87.1	-0.10 ± 0.03	198	-0.20 ± 0.01	735	$\mathbf{1.08 \pm 0.02}$	364
		JOR	0.00 ± 0.01	**76.5**	$\mathbf{-0.12 \pm 0.01}$	**177**	$\mathbf{-0.21 \pm 0.02}$	**626**	1.10 ± 0.02	**314**

where $\epsilon_{\text{align}} = 0.02$ (see sensitivity test in Appendix D.5). This tolerance accommodates small fluctuations in the estimated alignment metric while preventing structural modifications from substantially shifting the induced solution region away from the high-quality reference region under C_0. We note that R_{cw} is not intended as a formal test of distributional equivalence, but rather as an empirical safeguard for region-level alignment. To reduce reliance on this single statistic, we additionally report the nearest-neighbor ratio in Section 4.5 as a complementary local proximity metric, which yields qualitatively consistent conclusions.

4 Experiment

We evaluate JOR across four domains with distinct forms of guidance objectives: combinatorial optimization, LLM inference acceleration, Bayesian optimization, and reinforcement learning reward shaping. The detailed task objective and guidance objective of these tasks are presented in Appendix C. Across all tasks, the underlying solver or learner $A_{\mathcal{B}}$ and its computational budget are fixed. JOR only reshapes the guidance objective, while the original task objective C_0 remains unchanged and serves as the final evaluation criterion. Across all domains, structural proposals are generated and screened using the same JOR framework, followed by numerical optimization based on downstream performance. Detailed descriptions of the benchmarks, datasets, solver configurations, hyperparameters, evaluation protocols, and computational costs are provided in Appendix D.1. The resulting guidance objection function are provided in Appendix D.2.

4.1 Combinatorial Optimization

We evaluate JOR on CVRP using three routing backends, including LKH-3 (Helsgaun, 2000), NeuroLKH (Xin et al., 2021), and PyVRP (Wouda et al., 2024), across instances with 200-1,000 customers and search budgets ranging from 100 to 10,000 trials or iterations. To assess transferability, the guidance objective discovered with LKH-3 is directly transferred to NeuroLKH without retraining, and objectives discovered under a fixed search budget are further evaluated under different budgets. We compare JOR against the vanilla guidance objective and several LLM-based baselines, including Claude-Goal, ShinkaEvolve (Lange et al., 2026), and CORAL (Qu et al., 2026). We further integrate JOR with CORAL, yielding JOR-CORAL, to examine whether JOR can improve objectives produced by an existing structural-discovery framework.

As shown in Table 1 and Table 9, JOR improves performance in the large majority of configurations and achieves the best results in most settings. For example, with 1,000 LKH-3 trials, JOR reduces the

Table 2: LLM inference acceleration on six benchmarks. MAT (↑) is mean accepted tokens per decoding step; Speedup (↑) is relative to vanilla autoregressive decoding.

Task	EAGLE-3		TALON		Claude-goal		ShinkaEvolve		CORAL		JOR	
	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup	MAT	Speedup
MT-Bench	6.82	3.73±0.13	7.28	3.76±0.14	7.65	3.76±0.08	7.65	3.54±0.03	7.65	3.48±0.02	**7.75**	**3.79±0.27**
HumanEval	7.79	4.52±0.08	9.41	4.82±0.10	9.74	4.68±0.05	9.49	4.53±0.01	9.62	4.47±0.04	**10.00**	**4.87±0.19**
GSM8K	6.56	**3.59±0.19**	6.76	3.48±0.20	6.90	3.55±0.06	7.03	3.44±0.01	6.93	3.28±0.15	**7.15**	3.35±0.36
Alpaca	6.43	**3.64±0.18**	6.28	3.54±0.20	6.70	3.47±0.04	6.80	3.40±0.02	**6.83**	3.35±0.05	**6.83**	3.57±0.34
CNN/DM	6.63	3.50±0.08	7.19	3.54±0.12	7.25	3.39±0.03	7.33	3.31±0.01	7.31	3.27±0.02	**7.42**	**3.56±0.20**
NQ	5.15	3.10±0.02	4.94	3.08±0.02	5.09	2.82±0.01	5.21	2.77±0.02	**5.22**	2.78±0.01	5.20	**3.17±0.03**
Mean (480)	6.73	3.68±0.07	7.16	3.70±0.07	7.45	3.61±0.03	7.46	3.50±0.02	7.47	3.44±0.04	**7.60**	**3.72±0.13**

Table 3: Reference-region alignment under different shaping objectives. Cross/within values closer to 1 indicate better alignment. Lower NN ratios indicate stronger proximity to the reference region.

Domain	Vanilla		JOR		JOR$\lambda \times 3$		JOR$\lambda \times 10$		random ϕ	
	Cross/Within	NN Ratio	Cross/Within	NN Ratio	Cross/Within	NN Ratio	Cross/Within	NN Ratio	Cross/Within	NN Ratio
LKH	1.007±0.007	1.02	**1.002±0.007**	**1.00**	1.015±0.008	1.05	1.035±0.014	1.08	1.565±0.230	1.79
PyVRP	1.069±0.072	1.21	**1.056±0.063**	**1.17**	1.075±0.077	1.22	1.087±0.070	1.25	1.424±0.169	1.73
JSSP	1.074±0.083	1.54	**1.069±0.083**	**1.20**	1.075±0.091	1.27	1.102±0.103	1.66	1.098±0.103	1.64
EAGLE	1.511±0.033	2.22	**1.275±0.014**	**1.99**	1.447±0.022	5.08	1.481±0.022	5.56	3.872±0.069	25.9
BO	3.64 ± 1.71	3.33	**1.95 ± 1.12**	**1.28**	2.99 ± 1.33	2.70	3.14 ± 1.35	3.11	3.76 ± 1.69	3.49
RL	2.34 ± 0.21	3.70	**1.23 ± 0.09**	**0.53**	1.52 ± 0.02	0.66	1.81 ± 0.03	0.72	5.79 ± 0.70	13.6

optimality gaps on VRP200/500/1000 from $0.57\%/0.87\%/1.38\%$ to $0.53\%/0.65\%/1.09\%$, respectively. CORAL occasionally underperforms the vanilla guidance objective, suggesting that structural modification alone does not reliably translate into better downstream performance. In contrast, JOR-CORAL consistently improves upon CORAL, demonstrating that JOR can further refine objectives discovered by an existing structural adaptation method. Overall, JOR delivers robust gains across routing backends, problem scales, and search budgets, while also exhibiting cross-solver and cross-budget transfer. We additionally evaluate JOR on JSSP, with results reported in Appendix D.3.

4.2 LLM Inference Acceleration

We evaluate JOR on LLM inference acceleration with EAGLE-3, where the guidance objective controls the expansion strategy of the draft verification tree. During objective discovery, JOR optimizes the mean accepted draft tokens per decoding step (MAT), while end-to-end wall-clock speedup is evaluated separately. We compare against the original EAGLE-3 (Li et al., 2025b), TALON (Liu et al., 2026), and LLM-based baselines, including Claude-goal, ShinkaEvolve (Lange et al., 2026), and CORAL (Qu et al., 2026).

We evaluate on six representative LLM tasks: MT-Bench, HumanEval, GSM8K, Alpaca, CNN/DM, and Natural Questions (NQ). Table 2 reports MAT and end-to-end decoding speedup. JOR achieves the best average MAT and the highest or tied-highest MAT on five of the six benchmarks, increasing the average MAT from 6.73 for EAGLE-3 to 7.60. These results demonstrate that the reshaped expansion guidance improves draft-token acceptance. JOR also achieves the highest average wall-clock speedup of $3.72\times$, compared with $3.68\times$ for EAGLE-3 and $3.70\times$ for TALON. The wall-clock gains are modest and vary across tasks, with lower speedups than EAGLE-3 on GSM8K and Alpaca, as our main goal is to improve the MAT.

4.3 Bayesian Optimization

We evaluate JOR on Bayesian optimization (BO), where the guidance objective corresponds to the acquisition function that determines the next query point. We compare JOR with standard BO baselines, including EI (Jones et al., 1998), UCB ($\beta = 2$) (Srinivas et al., 2009), and MES (Wang & Jegelka, 2017), as well as LLM baseline, including Claude-goal, ShinkaEvolve (Lange et al., 2026), Eureka (Ma et al., 2024), α_{funbo} (Aglietti et al., 2024) and CORAL (Qu et al., 2026). Both methods use the same Claude backbone, controlling for the underlying language-model capability.

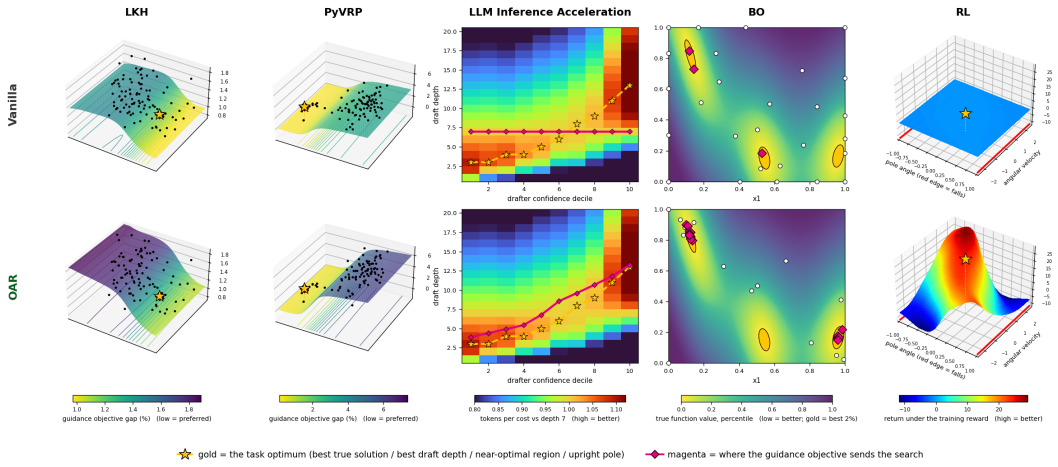

Figure 3: Landscape reshaping process under JOR across different problems.

We evaluate on four BO protocols: OOD-Bench, ID-Bench, few-shot Ackley-2D, and HPO-ID AdaBoost, covering synthetic black-box functions, distribution-shifted optimization, few-shot adaptation, and hyperparameter optimization. As shown in Table 7 in Appendix, JOR consistently achieves the lowest optimization cost across different evaluation budgets and protocols. These results show that JOR can effectively reshape acquisition objectives for Bayesian optimization.

4.4 Reinforcement Learning Reward Shaping

We evaluate JOR on reward shaping for reinforcement learning, where the guidance objective determines additional learning signals while the original environment reward remains unchanged for evaluation. We compare against standard PPO (Schulman et al., 2017), representative reward shaping methods (Hu et al., 2020) including NS, EM, and DPBA (Harutyunyan et al., 2015), as well as LLM baseline, including Claude-goal, ShinkaEvolve (Lange et al., 2026), Eureka (Ma et al., 2024), R* (Li et al., 2025a) and CORAL (Qu et al., 2026). Both CORAL and JOR use the same Claude backbone for discovering shaping functions, enabling a comparison of the optimization framework.

We evaluate on sparse CartPole with seven OOD physical variants and four training budgets (100k, 200k, 400k, and 1.2M environment steps). As shown in Table 8 in Appendix, JOR consistently achieves the highest ASPE across all budgets, improving from 172.7 at 100k steps to 197.6 at 1.2M steps. The advantage is especially significant in the early training stage, where effective shaping signals are more important under sparse feedback. These results demonstrate that JOR can automatically discover effective reward shaping objectives and generalize beyond combinatorial optimization to sequential decision-making tasks.

4.5 Analysis

Objective alignment analysis. We examine whether JOR improves downstream performance while preserving the solution region induced by the original task objective C_0. For analysis, we recompute the same metrics on held-out instances. For each instance, a high-quality reference region is constructed using a substantially stronger task-specific procedure under C_0, retaining all distinct outputs. We then compare Vanilla, JOR, coefficient-amplified variants ($3\times$ and $10\times$), and random shaping using cross/within and nearest-neighbor (NN) ratios, with metric definitions and domain-specific distances provided in Appendix D.4. As shown in Table 3, JOR consistently yields regions closer to the reference than Vanilla across all six domains, while coefficient amplification and random shaping generally increase deviation. These results indicate that JOR improves finite-budget performance while better preserving the task-aligned solution region, highlighting the importance of calibrating the numerical strength of structural modifications.

Table 4: Ablation study of structural-numerical coupling in JOR.

Variant	PyVRP			LLM Inference Acceleration		BO	RL
	200-Gap(%) ↓	500-Gap(%) ↓	1000-Gap(%) ↓	MAT ↑	Speedup ↑	Cost ↓	ASPE ↑
JOR (Full)	**4.03±0.16**	**3.83±0.02**	**3.20±0.03**	**7.60**	**3.72±0.13**	**0.798±0.056**	**172.7±22.6**
JOR w/o zero-order optimization	4.12±0.11	4.02±0.08	3.30±0.05	7.00	3.06±0.12	0.832±0.057	160.0±27.4
JOR w/o Historical Structural Memory	4.05±0.07	3.90±0.07	3.25±0.01	7.29	3.09±0.10	0.811±0.069	166.7±26.6
Edited-Component-Only Optimization	4.06±0.09	3.97±0.03	3.26±0.04	7.18	3.17±0.09	0.817±0.066	164.3±22.4
λ-Only Optimization	4.05±0.09	3.91±0.06	3.23±0.06	7.31	3.28±0.13	0.806±0.074	168.2±20.5
θ-Only Optimization	4.09±0.06	3.93±0.06	3.27±0.05	7.20	3.37±0.10	0.814±0.070	167.8±21.8

Landscape reshaping analysis. We further visualize how JOR reshapes the guidance landscape searched. Figure 3 compares Vanilla and JOR on fixed held-out instance. For LKH and PyVRP, JOR increases the separation between mediocre solutions and the task-optimal basin, producing a clearer descent toward high-quality solutions. For LLM inference acceleration, JOR adapts draft depth to model confidence and better follows the high-efficiency ridge, rather than using a fixed depth. For BO, JOR concentrates substantially more queries near the optimum (16/30 versus 3/30 within the best 2% region). For RL, JOR transforms the nearly flat sparse-reward landscape into a more informative landscape that favors high-return states. Across these settings, JOR preserves the task-optimal region while reshaping the guidance landscape to make search more effectively, consistent with the alignment analysis above.

Ablation study. We first conduct ablation studies of structural–numerical coupling on PyVRP, LLM inference acceleration, Bayesian optimization, and reinforcement learning, using 100-iteration routing gaps, MAT and speedup, OOD-Bench optimization cost, and 100k-step ASPE, respectively. As shown in Table 4, removing zeroth-order optimization or Historical Structural Memory consistently degrades performance, with a larger drop when numerical optimization is removed. Restricting numerical optimization to only the edited component, only λ, or only θ also underperforms full JOR, showing that structural changes can alter the calibration of previously active components. These results support jointly realigning the complete active numerical realization after structural adaptation, while structural memory further improves the evolution of guidance objectives.

We further ablate the objective-alignment safeguard by measuring the fraction of accepted candidates that become harmful under a longer evaluation budget. As shown in Table 11, removing objective alignment substantially increases this long-budget harm rate across all four domains: from 0% to 15.38% on COP PyVRP, 0% to 16.67% on LLM inference acceleration, 3.85% to 28.85% on BO, and 0% to 22.22% on RL. These results suggest that objective alignment helps suppress updates that appear beneficial during discovery but become detrimental under a different evaluation regime.

5 Conclusion

In this paper, we introduced *Joint Objective Reshaping (JOR)*, a framework for reshaping guidance objectives. JOR addresses two key challenges: *structural-numerical coupling*, where structural changes can invalidate previously calibrated parameters, and *objective alignment*, where optimizing downstream performance for a specific regime may lead the reshaped objective away from task-defined high-quality solution regions. By combining LLM-driven structural adaptation with zeroth-order optimization and an alignment safeguard, JOR jointly recalibrates all active numerical variables in a reparameterized latent space while preserving consistency with the original task objective. Across combinatorial optimization, LLM inference acceleration, Bayesian optimization, and reinforcement learning, JOR achieves broad performance improvements across different settings. Ablation and landscape analysis further support the importance of joint numerical optimization and objective alignment, showing that JOR improves the guidance landscape while largely preserving the high-quality solution region under the original task objective.

A limitation of JOR is that its alignment mechanism requires a task-specific procedure to construct a high-quality reference region during offline objective reshaping, introducing additional cost. More broadly, JOR assumes sufficiently frequent access to the task objective for numerical optimization and candidate evaluation, which may limit applicability when the task objective is unavailable or expensive to evaluate. Developing reference-free and more sample-efficient reshaping methods under limited task-objective feedback is an important direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing, improving the clarity of the manuscript. Beyond language editing, LLMs are also a core component of the proposed method: JOR uses an LLM-based coding agent as the structural proposer. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Virginia Aglietti, Ira Ktena, Jessica Schrouff, Eleni Sgouritsa, Francisco JR Ruiz, Alan Malek, Alexis Bellot, and Silvia Chiappa. Funbo: Discovering acquisition functions for bayesian optimization with funsearch. *arXiv preprint arXiv:2406.04824*, 2024.

Letian Chen, Nina Moorman, and Matthew Gombolay. Elemental: interactive learning from demonstrations and vision-language models for reward design in robotics. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

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

Xu Han, Qiannan Yang, Xianda Chen, Zhenghan Cai, Xiaowen Chu, and Meixin Zhu. Autoreward: Closed-loop reward design with large language models for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2024.

Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowé. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in neural information processing systems*, 33:15931–15941, 2020.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, pp. 74026–74078, 2026.

Mingjie Li, Jin-Kao Hao, and Qinghua Wu. Combining hash-based tabu search and frequent pattern mining for job-shop scheduling. *IISE Transactions*, 58(6):722–737, 2026.

Pengyi Li, Jianye Hao, Hongyao Tang, Yifu Yuan, Jinbin Qiao, Zibin Dong, and Yan Zheng. R*: Efficient reward design via reward structure evolution and parameter alignment optimization with large language models. In *ICML*, 2025a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2025b.

Tianyu Liu, Qitan Lv, Yuhao Shen, Xiao Sun, and Xiaoyan Sun. Talon: Confidence-aware speculative decoding with adaptive token trees. *arXiv preprint arXiv:2601.07353*, 2026.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, pp. 26516–26560, 2024.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Jonathan Sorg, Richard L Lewis, and Satinder Singh. Reward design via online gradient ascent. *Advances in Neural Information Processing Systems*, 23, 2010.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International conference on machine learning*, pp. 3627–3635. PMLR, 2017.

Dixiao Wei, Peng Yi, Jinlong Lei, Yiguang Hong, Hairong Dong, and Yuchuan Du. An automated reinforcement learning reward design framework with large language model for cooperative platoon coordination. *IEEE Transactions on Intelligent Transportation Systems*, 2026.

Niels A Wouda, Leon Lan, and Wouter Kool. Pyvrp: A high-performance vrp solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024.

Tianbao Xie, Siheng Zhao, Chen Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, pp. 35663–35699, 2024.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34:7472–7483, 2021.

Changwei Yao, Xinzi Liu, Chen Li, and Marios Savvides. Reward evolution with graph-of-thoughts: A bi-level language model framework for reinforcement learning. *arXiv preprint arXiv:2509.16136*, 2025.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *Advances in neural information processing systems*, 31, 2018.

Appendix

A Related Work **13**

B Methodology Details **13**

B.1 Why Joint Optimization is Necessary 13

B.2 Structural Adaptation Details 14

B.3 Zeroth-Order Optimization 15

C Task Definition **18**

C.1 Combinatorial Optimization 18

C.2 LLM Inference Acceleration 19

C.3 Bayesian Optimization 20

C.4 Reinforcement Learning Reward Shaping 21

D Additional Experimental Details **22**

D.1 Implementation Details 22

D.2 Discovered Guidance Objectives 24

D.3 The Results on JSSP 34

D.4 Reference-Region Alignment Metrics 35

D.5 Sensitivity to the Alignment Tolerance. 39

D.6 Applicability across Coding Agents 39

A Related Work

Numerical Reshaping. Early work on objective design mainly optimizes the numerical realization of a predefined structure (Sorg et al., 2010; Zheng et al., 2018; Hu et al., 2020). The online gradient-ascent approach (Sorg et al., 2010) formulates objective design as an optimization problem and adjusts objective parameters to improve the performance of a bounded agent under the designer's objective. LIRPG (Zheng et al., 2018) jointly learns policy parameters and the parameters of an intrinsic shaping function so that the resulting policy performs better under the extrinsic task objective. BiPaRS (Hu et al., 2020) assumes a given shaping objective and learns a parameterized shaping-weight function to control its influence on policy learning according to performance under the original task objective. Although these methods differ in how numerical variables are optimized, they share a common assumption that the functional form of the shaping objective is specified in advance, with adaptation performed within a fixed structural form and task setting.

Structural Reshaping. Recent work has moved beyond numerical realization by using LLMs to construct or revise executable guidance functions structure. Language-to-Rewards (Yu et al., 2023) translates natural-language instructions into guidance functions, which are then optimized by a downstream model-predictive controller to generate low-level robot actions, whereas Text2Reward (Xie et al., 2024) directly generates free-form dense objective code and allows further refinement through human feedback. Eureka (Ma et al., 2024) takes a more iterative approach, performing evolutionary search over LLM-generated objective programs and using downstream policy-training results as feedback for subsequent generations. Similar ideas have also been extended to the discovery of acquisition functions, search heuristics, and more general algorithmic programs (Aglietti et al., 2024; Lange et al., 2026; Qu et al., 2026). While these approaches substantially expand the space of possible structural modifications, they generally lack a principled mechanism for jointly adapting the numerical parameters after structural changes.

Structural–Numerical Reshaping. A few recent methods have begun to combine structural objective discovery with numerical adaptation, but their numerical adjustment mechanisms rely on task-specific forms of supervision. R* (Li et al., 2025a) evolves objective structures and refines their parameters through preference learning over trajectory segments. Rather than directly optimizing the task objective, it uses LLM-generated critics to produce pairwise trajectory preferences, limiting its applicability to settings where local trajectories can be meaningfully compared. ELEMENTAL (Chen et al., 2025) constructs objective features from language and visual demonstrations, and estimates their weights through inverse reinforcement learning, requiring an appropriate visual observation interface. These works require task-dependent feedback representations and are therefore difficult to transfer across heterogeneous optimization settings. JOR instead uses downstream performance under the original task objective as a common optimization signal for both structural selection and joint numerical refinement, without requiring trajectory-level preferences or demonstration-based supervision. This provides a unified structural–numerical optimization framework applicable across diverse guidance objectives and downstream algorithms.

B Methodology Details

B.1 Why Joint Optimization is Necessary

A structural edit can change the effective contribution of previously active components, making their previous numerical configuration suboptimal. The following result formalizes why optimizing only the edited component may be insufficient after a structural change.

Proposition 1 (Necessity of Joint Numerical Optimization)**.** *Consider a screened candidate structure $\widetilde{\Phi}_T$ at outer iteration T. Let $\xi = (\xi_e, \xi_i)$ denote the latent numerical variables associated with the edited and incumbent components, respectively. Let*

$$\widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \xi)$$

denote the Gaussian-smoothed empirical downstream objective defined in Appendix B.3, where $\mathcal{Z}$ is the fixed evaluation set used for zeroth-order numerical optimization.

Suppose optimization restricted to the edited variables reaches a point $\tilde{\xi}$ satisfying

$$\nabla_{\xi_e} \widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \tilde{\xi}) = 0.$$

If

$$\nabla_{\xi_i} \widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \tilde{\xi}) \neq 0,$$

then $\tilde{\xi}$ is not stationary for the full numerical optimization problem, and a descent direction exists by adjusting the incumbent numerical variables.

Proof. Under the stated conditions,

$$\nabla_{\xi} \widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \tilde{\xi}) = \begin{bmatrix} 0 \\ \nabla_{\xi_i} \widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \tilde{\xi}) \end{bmatrix} \neq 0.$$

Consider the direction

$$d = \begin{bmatrix} 0 \\ -\nabla_{\xi_i} \widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \tilde{\xi}) \end{bmatrix}.$$

Its directional derivative is

$$\nabla_{\xi} \widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \tilde{\xi})^\top d = -\left\| \nabla_{\xi_i} \widehat{J}_{\mathcal{Z},\sigma}(\widetilde{\Phi}_T, \tilde{\xi}) \right\|_2^2 < 0.$$

Hence, d is a descent direction for the Gaussian-smoothed empirical downstream objective. Therefore, optimizing only the edited numerical variables is insufficient whenever a nonzero gradient remains with respect to the incumbent variables. □

B.2 Structural Adaptation Details

JOR maintains a lightweight structural memory

$$\mathcal{M}_T = (\mathcal{K}_T, \mathcal{P}_T, \mathcal{A}_T, \mathcal{R}_T), \tag{10}$$

where $\mathcal{K}_T$ indexes the active shaping components in the current accepted structure

$$\Phi_T = \{\phi_k : k \in \mathcal{K}_T\},$$

$\mathcal{P}_T$ denotes the Candidate Pool of previously screened but inactive proposals, $\mathcal{A}_T$ is the Rejected Archive containing discarded or superseded proposals together with their failure evidence, and $\mathcal{R}_T$ records downstream performance across evaluation cells. All performance records are evaluated under the original task objective C_0 relative to the fixed base guidance objective G_0. Given the operator set $\mathcal{O} = \{\mathrm{ADD}, \mathrm{UPDATE}, \mathrm{DROP}, \mathrm{REPLACE}\}$, the structural proposer generates a set of candidate edits

$$\mathcal{E}_T = \bigcup_{o \in \mathcal{O}} \Pi^{(o)}(\mathcal{M}_T) = \{e_T^{(j)}\}_{j=1}^{N_T}, \tag{11}$$

where N_T is the number of structural proposals generated at outer iteration T. The proposal operator $\Pi^{(o)}$ is implemented by the LLM in our experiments, although the framework does not require a specific proposal mechanism. The proposer only generates candidate structural modifications; their utility is determined by downstream evaluation under C_0.

Applying an edit $e_T^{(j)}$ to the current structure Φ_T produces a candidate structure $\widetilde{\Phi}_T^{(j)}$. For ADD, UPDATE, and REPLACE proposals, the proposer additionally specifies executable component definitions, tunable numerical variables, and their admissible ranges. Before joint numerical optimization, only the newly introduced or modified component is provisionally calibrated, while the numerical variables of the remaining active components are inherited from the current configuration. This yields an initial numerical configuration

$$\bar{\eta}_j = \textsc{InitNum}\left(e_T^{(j)}, \eta_T\right) \tag{12}$$

for candidate $\widetilde{\Phi}_T^{(j)}$. For DROP proposals, the selected component and its associated numerical variables are removed, while all remaining numerical variables are kept unchanged. Thus, $\bar{\eta}_j$ is precisely the fixed numerical center used by the latent-space parameterization in Sec. 3.2 and Appendix B.3.

The screening stage is intentionally conservative. It removes candidates that fail executable compatibility, computational-budget constraints, coarse downstream feasibility, activity, or redundancy checks. Let

$$\left\{\left(\widetilde{\Phi}_T^{(j)}, \bar{\eta}_j\right)\right\}_{j=1}^{N_T'}$$

denote the $N'_T \leq N_T$ candidates retained after screening. Final candidate selection is not performed at this stage. For redundancy screening of a candidate edit e, let $o(e)$ denote its edit operator and define the excluded target set

$$\mathcal{X}(e) = \begin{cases} \emptyset, & o(e) = \text{ADD}, \\ \{\text{target}(e)\}, & o(e) \in \{\text{UPDATE}, \text{REPLACE}\}. \end{cases} \tag{13}$$

The remaining active shaping signal is

$$S^{\text{bank}}_{T,-e}(v; x) = \sum_{k \in \mathcal{K}_T \setminus \mathcal{X}(e)} \lambda_k \phi_k(v; x; \theta_k), \tag{14}$$

and redundancy is measured by

$$r_{\text{scr}}(e) = \left| \text{corr}\left(\phi_e(\cdot; \bar{\theta}_e), S^{\text{bank}}_{T,-e}\right) \right|, \tag{15}$$

where $\bar{\theta}_e$ denotes the provisionally initialized parameters of the edited component. For UPDATE and REPLACE, excluding the targeted component prevents a revised mechanism from being rejected merely because it remains similar to its previous version. After screening, each retained pair $(\widetilde{\Phi}^{(j)}_T, \bar{\eta}_j)$ is passed to the joint numerical optimization stage. The structural memory is updated only after numerical optimization and objective-aligned acceptance: components in the accepted configuration update the active Bank $\mathcal{K}_{T+1}$, screened but inactive proposals may be retained in $\mathcal{P}_{T+1}$, and rejected or superseded proposals together with their downstream outcomes are recorded in $\mathcal{A}_{T+1}$ and $\mathcal{R}_{T+1}$.

B.3 Zeroth-Order Optimization

This section provides the derivation and implementation details of zeroth-order numerical optimization. At outer iteration T, consider one screened candidate

$$\left(\widetilde{\Phi}^{(j)}_T, \bar{\eta}_j\right),$$

where $\bar{\eta}_j$ denotes the numerical configuration obtained after structural screening. For notational simplicity, we omit the candidate index j throughout this section and write

$$\Phi := \widetilde{\Phi}^{(j)}_T, \qquad \bar{\eta} := \bar{\eta}_j.$$

Both the candidate structure Φ and the numerical center $\bar{\eta}$ remain fixed throughout the following numerical optimization.

Latent Space Reparameterization. Zeroth-order optimization jointly optimizes the numerical realization $\eta = (\theta, \lambda)$ of all active shaping components. Direct isotropic perturbation in the original numerical space can be inappropriate because different variables may have heterogeneous scales and admissible ranges. We therefore perform zeroth-order perturbations in an unconstrained latent space. Let $\mathcal{H}(\Phi)$ denote the admissible numerical space associated with the fixed candidate structure Φ. We define a validity-preserving mapping

$$\mathcal{T} : \mathbb{R}^{d_\eta} \to \mathcal{H}(\Phi), \qquad \eta = \mathcal{T}(\bar{\eta}, \xi),$$

where $\xi \in \mathbb{R}^{d_\eta}$ is the latent variable. For a variable with a symmetric valid local optimization range, we use

$$\eta_k(\xi_k) = \bar{\eta}_k + b_k \tanh(\xi_k), \qquad k = 1, \ldots, d_\eta, \tag{16}$$

where $b_k > 0$ determines the maximum local deviation of the k-th numerical variable from $\bar{\eta}_k$.

For asymmetric or one-sided admissible ranges, the mapping can be extended using separate lower and upper radii. For example, for the valid interval

$$[\bar{\eta}_k - b_k^-, \bar{\eta}_k + b_k^+],$$

a validity-preserving mapping is

$$\eta_k(\xi_k) = \bar{\eta}_k + \begin{cases} b_k^+ \tanh(\xi_k), & \xi_k \geq 0, \\ b_k^- \tanh(\xi_k), & \xi_k < 0. \end{cases}$$

This preserves $\eta_k(0) = \bar{\eta}_k$ and reduces to the symmetric parameterization when $b_k^- = b_k^+$. The resulting local range is verified to contain only admissible numerical configurations; otherwise, the corresponding bound is tightened or a task-specific validity-preserving mapping is used. Numerical optimization is initialized at $\xi_0 = 0$, such that $\mathcal{T}(\bar{\eta}, \xi_0) = \bar{\eta}$. During the inner numerical optimization, $\bar{\eta}$ remains fixed, whereas the latent iterate ξ_t is updated. The corresponding numerical configuration at step t is $\eta_t = \mathcal{T}(\bar{\eta}, \xi_t)$. Thus, Gaussian perturbations are isotropic in latent space, but not necessarily in the original numerical space.

Fixed Empirical Downstream Objective. The conceptual downstream objective in the main text is

$$J(\Phi, \eta) = \mathbb{E}_{x,\omega}\left[C_0\left(A_{\mathcal{B}}(G_{\Phi,\eta}; x, \omega); x\right)\right].$$

In practice, each numerical optimization uses a fixed evaluation set

$$\mathcal{Z} = \{(x_r, \omega_r)\}_{r=1}^{|\mathcal{Z}|}$$

of task instances and solver or learner randomness. We therefore optimize the empirical downstream objective

$$\widehat{J}_{\mathcal{Z}}(\Phi, \xi) = \frac{1}{|\mathcal{Z}|} \sum_{(x,\omega)\in\mathcal{Z}} C_0\left(A_{\mathcal{B}}\left(G_{\Phi,\mathcal{T}(\bar{\eta},\xi)}; x, \omega\right); x\right). \tag{17}$$

The same evaluation set $\mathcal{Z}$ is kept fixed across the local zeroth-order updates; the analysis below therefore concerns this empirical objective rather than an unobserved population objective.

Gaussian-Smoothed Objective. Because $\widehat{J}_{\mathcal{Z}}$ can be non-smooth or discontinuous due to discrete and budget-truncated downstream dynamics, we optimize its Gaussian-smoothed counterpart:

$$\widehat{J}_{\mathcal{Z},\sigma}(\Phi, \xi) = \mathbb{E}_{u\sim\mathcal{N}(0,I)}\left[\widehat{J}_{\mathcal{Z}}(\Phi, \xi + \sigma u)\right], \tag{18}$$

where $\sigma > 0$ is the latent-space smoothing scale.

Using the Gaussian smoothing identity (Nesterov & Spokoiny, 2017),

$$\nabla_\xi \widehat{J}_{\mathcal{Z},\sigma}(\Phi, \xi) = \frac{1}{\sigma}\mathbb{E}_u\left[\widehat{J}_{\mathcal{Z}}(\Phi, \xi + \sigma u)u\right]. \tag{19}$$

Thus, gradient information can be estimated using only downstream objective evaluations without differentiating through the solver or learner.

Antithetic Zeroth-Order Estimator. Given independent perturbation directions

$$u_i \sim \mathcal{N}(0, I), \qquad i = 1, \ldots, m,$$

the ideal antithetic estimator is

$$\widehat{g}(\xi) = \frac{1}{2m\sigma}\sum_{i=1}^{m}\left[\widehat{J}_{\mathcal{Z}}(\Phi, \xi + \sigma u_i) - \widehat{J}_{\mathcal{Z}}(\Phi, \xi - \sigma u_i)\right]u_i. \tag{20}$$

For the fixed empirical objective, this estimator is unbiased for the gradient of the Gaussian-smoothed objective:

$$\mathbb{E}\left[\widehat{g}(\xi)\right] = \nabla_\xi \widehat{J}_{\mathcal{Z},\sigma}(\Phi, \xi). \tag{21}$$

Joint Optimization of Active Components. For the fixed candidate structure Φ, the mapping $\mathcal{T}(\bar{\eta}, \xi)$ covers the complete active numerical realization $\eta = (\theta, \lambda)$. Hence, each perturbation jointly changes the numerical variables of all active components, rather than only those associated with the newly introduced or modified component. This is important because a structural modification can change the optimal numerical realization. In general, $\eta^\star(\Phi') \neq \eta^\star(\Phi)$. Zeroth-order optimization therefore jointly optimizes the complete active numerical realization after structural adaptation.

Stochastic Evaluation and Common Random Numbers. When the downstream algorithm is stochastic, the positive and negative perturbations in each antithetic pair are evaluated using the same task instances and solver or learner randomness contained in $\mathcal{Z}$:

$$\widehat{J}_{\mathcal{Z}}(\Phi, \xi + \sigma u_i) - \widehat{J}_{\mathcal{Z}}(\Phi, \xi - \sigma u_i). \tag{22}$$

This common-random-number construction preserves the marginal evaluation distribution while reducing variance when the paired evaluations are positively correlated.

Paired-Difference Clipping. Define the paired empirical difference

$$\Delta_\xi(u) = \widehat{J}_{\mathcal{Z}}(\Phi, \xi + \sigma u) - \widehat{J}_{\mathcal{Z}}(\Phi, \xi - \sigma u). \tag{23}$$

Discrete finite-budget algorithms can occasionally produce large downstream jumps under small perturbations. To limit the influence of such events, we clip the paired difference:

$$\Delta_\xi^\tau(u) = \mathrm{clip}\left(\Delta_\xi(u), -\tau, \tau\right). \tag{24}$$

The practical estimator is therefore

$$g_\tau(\xi) = \frac{1}{2m\sigma} \sum_{i=1}^{m} \Delta_\xi^\tau(u_i) u_i. \tag{25}$$

Clipping generally introduces bias. Let

$$g_\sigma(\xi) = \nabla_\xi \widehat{J}_{\mathcal{Z},\sigma}(\Phi, \xi). \tag{26}$$

Then

$$\mathbb{E}\left[g_\tau(\xi)\right] = g_\sigma(\xi) + b_\tau(\xi), \tag{27}$$

where

$$b_\tau(\xi) = \frac{1}{2\sigma} \mathbb{E}_u\left[\left(\mathrm{clip}(\Delta_\xi(u), -\tau, \tau) - \Delta_\xi(u)\right) u\right]. \tag{28}$$

For d_η latent numerical variables, Cauchy–Schwarz gives

$$\|b_\tau(\xi)\| \le \frac{\sqrt{d_\eta}}{2\sigma} \left(\mathbb{E}_u\left[\left(|\Delta_\xi(u)| - \tau\right)_+^2\right]\right)^{1/2}. \tag{29}$$

Thus, paired-difference clipping trades unbiasedness for bounded sensitivity to unusually large paired objective differences. We do not assume that this bias is negligible or claim monotonic improvement of individual zeroth-order updates.

Numerical Update, Raw Replay, and Final Gate. For numerical optimization steps $t = 0, \ldots, t_{\max} - 1$, JOR updates the latent iterate using Adam:

$$\xi_{t+1} = \mathrm{Adam}\left(\xi_t, g_\tau(\xi_t)\right), \tag{30}$$

and obtains the corresponding numerical configuration through

$$\eta_{t+1} = \mathcal{T}(\bar{\eta}, \xi_{t+1}). \tag{31}$$

After $t_{\max}$ numerical optimization steps, the resulting numerical movement is evaluated using raw downstream performance rather than the Gaussian-smoothed surrogate. Let $\mathcal{Z}_{\mathrm{replay}}$ denote an independent replay set and $\xi_{t_{\max}}$ the final latent iterate. The numerical movement is retained only when

$$\widehat{J}_{\mathcal{Z}_{\mathrm{replay}}}(\Phi, \xi_{t_{\max}}) - \widehat{J}_{\mathcal{Z}_{\mathrm{replay}}}(\Phi, \xi_0) \le -\delta_{\mathrm{replay}}, \tag{32}$$

where $\delta_{\mathrm{replay}} \ge 0$ is the replay margin. Otherwise, the numerical configuration reverts to its pre-optimization value $\bar{\eta}$.

The replay step only determines whether the local numerical update is retained. Final acceptance of the complete candidate configuration remains governed by the objective-aligned acceptance criterion in Eq. (9). Hence, improvement of the Gaussian-smoothed empirical objective is not assumed to imply improvement of the raw downstream objective or final acceptance of the structural candidate.

Gaussian smoothing and antithetic zeroth-order estimation are standard tools. The role of zeroth-order optimization in JOR is to combine them with bounded latent-space perturbations, joint optimization of all active numerical variables, paired stochastic evaluation, stabilization, and downstream validation after structural adaptation.

C TASK DEFINITION

JOR is evaluated on diverse optimization and decision-making problems to examine whether objective reshaping can improve different forms of intermediate guidance objectives while keeping the final task objective unchanged. Across all tasks, the underlying solver or learner is frozen under a fixed computational budget. JOR only modifies the guidance signal consumed by the solver or learner, while all evaluations are performed using the original task objective.

Following the formulation in Sec. 3.1, the reshaped guidance objective takes the general form

$$G_{\Phi,\eta}(v;x) = G_0(v;x) + \sum_{k=1}^{K} \lambda_k \phi_k(v;x;\theta_k), \qquad \eta = (\theta, \lambda), \tag{33}$$

where v denotes the decision context at which the guidance objective is evaluated. Depending on the task, v may represent an edge or local search move in combinatorial optimization, a tree node in speculative decoding, a candidate query in Bayesian optimization, or a transition in reinforcement learning.

C.1 COMBINATORIAL OPTIMIZATION

We evaluate JOR on two classical combinatorial optimization problems: capacitated vehicle routing and job-shop scheduling. Although both problems use heuristic solvers, the guidance objective is instantiated differently according to the solver interface.

CVRP. The Capacitated Vehicle Routing Problem (CVRP) aims to construct a set of vehicle routes satisfying capacity constraints while minimizing total travel distance.

Given a depot and a set of customers,

$$V = \{0, 1, \ldots, n\}, \tag{34}$$

where node 0 denotes the depot, customer i has demand ρ_i, and each vehicle has capacity Q. Let

$$\mathbf{W} = [w_{ij}] \tag{35}$$

denote the original travel-distance matrix. A routing solution is represented as

$$s = \{r_1, \ldots, r_{n_r}\}, \tag{36}$$

where each route satisfies

$$\sum_{i \in r_h} \rho_i \le Q, \qquad h = 1, \ldots, n_r. \tag{37}$$

The original task objective is the total travel distance:

$$C_0(s;x) = \sum_{r \in s} \sum_{(i,j) \in r} w_{ij}. \tag{38}$$

The frozen routing solver operates under the reshaped guidance objective:

$$s = A_{\mathcal{B}}\left(G_{\Phi,\eta}; x, \omega\right), \tag{39}$$

while the resulting solution is evaluated using the task objective C_0. The base guidance objective G_0 corresponds to the original routing guidance derived from the travel-distance costs. We consider two routing solvers: LKH-3 (Helsgaun, 2000) and PyVRP (Wouda et al., 2024). For both solvers, the computational budget and random seeds are fixed between the baseline and reshaped guidance objectives.

JSSP. The Job-Shop Scheduling Problem (JSSP) considers scheduling a set of jobs on multiple machines while minimizing the overall completion time.

Given n_J jobs and n_M machines, job i consists of an ordered sequence of operations

$$\mathcal{O}_i = \left(O_{i1}, O_{i2}, \ldots, O_{in_M}\right), \tag{40}$$

where operation $O_{i\ell}$ requires a specific machine and has processing time $p_{i\ell}$.

A schedule s assigns a start time to every operation. Let $\mathrm{st}_{i\ell}$ denote the start time of operation $O_{i\ell}$. The precedence constraints require

$$\mathrm{st}_{i,\ell+1} \geq \mathrm{st}_{i\ell} + p_{i\ell}, \tag{41}$$

together with the machine-capacity constraints.

The original task objective is the makespan:

$$C_0(s; x) = \max_i \left(\mathrm{st}_{i,n_M} + p_{i,n_M}\right). \tag{42}$$

Unlike CVRP, the JSSP solver directly evaluates local search moves. Therefore, the guidance objective is reshaped at the move-evaluation level. For a candidate local search move v, $G_0(v; x)$ denotes the original move-evaluation score, and JOR modifies it according to

$$G_{\Phi,\eta}(v; x) = G_0(v; x) + \sum_{k=1}^{K} \lambda_k \phi_k(v; x; \theta_k). \tag{43}$$

The solver selects moves according to the reshaped guidance objective, while the quality of the resulting schedule is evaluated only by the original makespan objective C_0.

The JSSP solver is a fixed-budget tabu-search implementation with deterministic iteration limits.

C.2 LLM Inference Acceleration

We further evaluate JOR on speculative decoding with EAGLE-3 (Li et al., 2025b). Here, x denotes a decoding instance. Given a target model and a draft model, speculative decoding constructs a verification tree whose nodes correspond to draft-token candidates.

EAGLE-3 ranks draft nodes using cumulative draft-model log probability:

$$\mathrm{CLP}(v; x) = \mathrm{CLP}(\mathrm{parent}(v); x) + \log p_{\mathrm{draft}}\left(v \mid \mathrm{parent}(v), x\right). \tag{44}$$

In the stock configuration used in our experiments, the top K_{beam} beam nodes are expanded up to a fixed depth of seven layers, and the final verification candidates are selected according to the same cumulative-log-probability ranking.

Importantly, JOR does *not* reshape this ranking score. Instead, following Talon (Liu et al., 2026), we represent the stock fixed-depth expansion policy using the base expansion guidance

$$G_0(v, h; x) = \begin{cases} +1, & h \leq 7, \\ -1, & h > 7, \end{cases} \tag{45}$$

where h denotes the depth of node v. With an empty shaping Bank, thresholding $G_0(v, h; x)$ at zero exactly reproduces the stock depth-7 expansion rule.

JOR reshapes only this expansion guidance:

$$G_{\Phi,\eta}(v, h; x) = G_0(v, h; x) + \sum_{k=1}^{K} \lambda_k \phi_k\left(\mathrm{ctx}(v, h; x); \theta_k\right). \tag{46}$$

A candidate node is expanded according to

$$\mathrm{Expand}(v, h; x) = \mathbb{I}\left[G_{\Phi,\eta}(v, h; x) \geq 0\right]. \tag{47}$$

Thus, JOR changes which nodes are expanded and how deep the draft tree is allowed to grow, while leaving the draft probabilities, cumulative-log-probability ranking, and final verification procedure unchanged.

The downstream task objective is the mean number of accepted draft tokens per decoding step (MAT). Following the minimization convention in Eq. 2, we define

$$C_0(s;x) = -\mathrm{MAT}(s;x) = -\frac{1}{S_{\mathrm{dec}}}\sum_{r=1}^{S_{\mathrm{dec}}} a_r, \tag{48}$$

where a_r denotes the number of accepted draft tokens at decoding step r and S_{dec} is the total number of decoding steps. Thus, minimizing C_0 corresponds to maximizing MAT. Structural selection and zeroth-order numerical optimization are both governed by this task objective, while wall-clock speedup is reported separately as an end-to-end efficiency metric.

C.3 Bayesian Optimization

Bayesian optimization (BO) considers expensive black-box optimization problems. In our general formulation, the task instance x specifies a black-box function f_x defined over a query domain $\mathcal{Y}$. The corresponding optimum is

$$y^\star = \arg\min_{y\in\mathcal{Y}} f_x(y). \tag{49}$$

At BO iteration ℓ, let

$$\mathcal{H}_\ell = \{(y_r, f_x(y_r))\}_{r=0}^{\ell}$$

denote the observation history. The surrogate model provides a predictive mean $\mu_\ell(y)$ and predictive uncertainty $\sigma_{\mathrm{GP},\ell}(y)$. The baseline acquisition function is expected improvement (EI).

For a candidate query y, define the guidance context as

$$v = (y, \mathcal{H}_\ell).$$

The base guidance objective is

$$G_0(v;x) = \mathrm{EI}\left(y \mid \mathcal{H}_\ell\right). \tag{50}$$

JOR does not modify the Gaussian-process surrogate or the black-box function f_x. Instead, it reshapes only the acquisition guidance:

$$G_{\Phi,\eta}(v;x) = G_0(v;x) + \sum_{k=1}^{K} \lambda_k \phi_k(v;x;\theta_k), \qquad \eta = (\theta,\lambda). \tag{51}$$

The next query is selected according to

$$y_{\ell+1} = \arg\max_{y\in\mathcal{Y}} G_{\Phi,\eta}\left((y,\mathcal{H}_\ell);x\right), \tag{52}$$

and is always evaluated using the original black-box function f_x.

Let L_{BO} denote the finite query budget. The resulting query trajectory is

$$s = (y_0, y_1, \ldots, y_{L_{\mathrm{BO}}}) = A_{\mathcal{B}}\left(G_{\Phi,\eta};x,\omega\right), \tag{53}$$

where L_{BO} is the task-specific realization of the computational budget $\mathcal{B}$, following FunBO (Aglietti et al., 2024).

The initial query y_0 is fixed by the benchmark protocol and is chosen as the worst candidate on the finite evaluation grid. This initialization is shared by all methods and is observed before the first acquisition step, so it does not provide the optimizer with additional access to f_x.

Let

$$f_x^\star = \min_{y\in\mathcal{Y}} f_x(y)$$

denote the grid optimum. For every non-constant f_x, $f_x(y_0) > f_x^\star$. We define the task objective as

$$\begin{aligned} C_0(s;x) &= \underbrace{\frac{\min_{0\le\ell\le L_{\mathrm{BO}}} f_x(y_\ell) - f_x^\star}{f_x(y_0) - f_x^\star}}_{\mathrm{Reg}_{\mathrm{BO}}} + \underbrace{\frac{\ell_{\mathrm{hit}}}{L_{\mathrm{BO}}}}_{\mathrm{Hit}_{\mathrm{BO}}}, \\ \ell_{\mathrm{hit}} &= \min\left\{\ell \in \{1,\ldots,L_{\mathrm{BO}}\} : f_x(y_\ell) = f_x^\star\right\}. \end{aligned} \tag{54}$$

If the optimum is never reached, $\ell_{\mathrm{hit}} = L_{\mathrm{BO}}$. Here $\mathrm{Reg}_{\mathrm{BO}}$ is the normalized simple regret and $\mathrm{Hit}_{\mathrm{BO}}$ is the normalized hitting time. Since the incumbent includes y_0 and y_0 is the grid maximum, $\mathrm{Reg}_{\mathrm{BO}} \in [0,1]$, and hence $C_0 \in [0,2]$. Lower is better.

C.4 Reinforcement Learning Reward Shaping

Finally, we evaluate JOR on sparse-reward reinforcement learning. Let π denote a policy. To avoid overloading the downstream objective $J(\Phi, \eta)$ defined in Sec. 3.1, we denote the standard reinforcement learning return by

$$\mathcal{J}_{\mathrm{RL}}(\pi) = \mathbb{E}_{\pi}\left[\sum_{\nu=0}^{\infty} \gamma^{\nu} r_{\nu}\right]. \tag{55}$$

We use the sparse CartPole environment following reward-shaping studies such as BiPaRS (Hu et al., 2020). Here, the task instance x specifies a physics configuration. Let o_{ν} denote the environment observation, a_{ν} the action, and r_{ν} the original sparse reward at environment step ν:

$$r_{\nu} = \begin{cases} -1, & \text{failure}, \\ 0, & \text{otherwise}. \end{cases} \tag{56}$$

For the transition-level guidance context

$$v_{\nu} = (o_{\nu}, a_{\nu}, o_{\nu+1}),$$

the base guidance objective corresponds to the original learning reward,

$$G_0(v_{\nu}; x) = r_{\nu}.$$

During training, JOR reshapes only this learning signal:

$$G_{\Phi,\eta}(v_{\nu}; x) = r_{\nu} + \sum_{k=1}^{K} \lambda_k \phi_k(v_{\nu}; x; \theta_k). \tag{57}$$

Equivalently, $G_{\Phi,\eta}(v_{\nu}; x)$ is the shaped reward supplied to PPO, while the original sparse reward in Eq. 56 remains unchanged for task evaluation.

The PPO optimizer and environment dynamics are fixed. Under a computational budget of $\mathcal{B}$ environment steps, the learner produces a training run

$$s = A_{\mathcal{B}}\left(G_{\Phi,\eta}; x, \omega\right), \tag{58}$$

where ω denotes training randomness.

Let

$$U_{\mathrm{PPO}} = \frac{\mathcal{B}}{20{,}000}$$

denote the number of policy updates. After each update $c = 1, \ldots, U_{\mathrm{PPO}}$, the deterministic policy π_c is evaluated for 20 episodes using only the original sparse reward in Eq. 56. Let $\mathrm{ASPE}_c \in [0, 200]$ denote the corresponding average steps per episode.

The task objective is the learning-curve ASPE:

$$C_0(s; x) = \frac{1}{U_{\mathrm{PPO}}} \sum_{c=1}^{U_{\mathrm{PPO}}} \mathrm{ASPE}_c. \tag{59}$$

This objective is maximized; equivalently, the minimization formulation in Eq. 2 can be applied to $-C_0$. The values reported in the experimental tables are C_0 itself, while $\mathrm{ASPE}_{U_{\mathrm{PPO}}}$ is additionally reported as the end-of-training performance.

The shaped guidance $G_{\Phi,\eta}$ is used only during PPO updates. Numerical optimization, candidate acceptance, and all reported evaluations are based on the task objective C_0, which is computed from evaluations under the original sparse reward.

D Additional Experimental Details

This section provides additional implementation details, datasets, and hyperparameter settings for all experiments. Unless otherwise specified, all experiments follow the same principle: the underlying solver or learner is frozen, JOR only modifies the guidance objective, and the final evaluation is always performed using the original task objective.

D.1 Implementation Details

All experiments are conducted on servers equipped with NVIDIA RTX PRO 6000 Blackwell GPUs and AMD EPYC 9555 CPUs. Different tasks use different subsets of the available GPUs and CPU cores according to their computational requirements; the task-specific resource allocation is reported in the corresponding experimental settings below. Table 5 summarizes the structural-search budgets and the main numerical hyperparameters used by zeroth-order optimization in each domain.

Table 5: Structural-search budgets and JZR hyperparameters used across domains.

Setting	**LKH-CVRP**	**PyVRP-CVRP**	**JSSP**	**EAGLE**	**BO**	**RL**
Structural rounds	40	40	40	40	20	20
Proposals / round	10	10	10	10	10	10
Candidates forwarded / round	3	3	3	3	3	3
Directions m	4	4	4	4	4	4
Batch size B	24	24	24	24	24	24
JZR steps R	8	8	8	8	8	8
Smoothing scale σ	0.3	0.3	0.3	0.3	0.3	0.3
Clipping threshold τ	0.05	0.05	0.05	0.05	0.05	0.05
Adam learning rate	$0.2 \to 0.04$	$0.2 \to 0.04$	$0.2 \to 0.04$	$0.2 \to 0.04$	$0.2 \to 0.04$	$0.2 \to 0.04$
Warmup steps	3	3	3	3	3	3

The structural proposer in JOR is implemented using Claude Code with Claude Sonnet 5 unless otherwise specified. Each proposal call is executed as an independent session and receives only the structural memory exposed by JOR, including the active Bank, Candidate Pool, Rejected Archive, and cell-level downstream diagnostics. The proposer has no access to held-out test instances or task-oracle information. For the alignment gate, $\mathcal{D}_{\text{align}}$ is drawn only from objective-discovery data and never from the final test set. Strong-reference outputs on $\mathcal{D}_{\text{align}}$ are precomputed or evaluated offline and are consumed only by the acceptance criterion; they are not exposed to the structural proposer or zeroth-order optimization.

The implementation combines Python-based orchestration with the native solver or learning backend used in each domain. LKH and JSSP tabu-search solvers are executed through compiled native binaries, PyVRP is invoked in process through its Python interface, EAGLE-3 uses PyTorch/CUDA with a Triton implementation for the final shaped expansion rule, Bayesian optimization is evaluated with parallel CPU workers, and the reinforcement-learning experiments are implemented in JAX and executed on GPU. All experiments are independently repeated three times, and we report aggregated results unless otherwise specified. Detailed software versions, solver configurations, computational budgets, and task-specific hardware allocations are reported in the following subsections.

CVRP. We evaluate JOR on the Capacitated Vehicle Routing Problem (CVRP) using two independent routing solvers: LKH-3 (Helsgaun, 2000) and PyVRP (Wouda et al., 2024). Training instances are generated following Uchoa-style CVRP distributions, including clustered, random, and mixed customer layouts. Instances contain different scales: $n \in \{100, 200, 500, 1000\}$. The training set contains 12 cells defined by the combination of instance scale and spatial distribution. Each cell contains 45 instances, where 30 are used for screening and 15 are reserved for numerical optimization. The final evaluation is performed on both synthetic and public benchmarks. The synthetic test set contains VRP instances with sizes 200, 500, and 1000, with 128 instances per scale. We additionally evaluate on CVRPLib X (Uchoa et al., 2017) instances covering sizes from 100 to 1001 nodes.

The routing solvers are executed under fixed computational budgets. For LKH, the budget is defined by MAX_TRIALS, while PyVRP uses a fixed iteration budget. The iteration budgets used during

training of LKH is 400, and PyVRP is 2000. The original routing objective is total travel distance as shown in Eq. 38. And JOR modifies only the internal guidance cost used by the solver and does not change feasibility constraints or final distance evaluation.

JSSP. We evaluate JOR on the Job-Shop Scheduling Problem using classical scheduling benchmarks (Demirkol et al., 1998). The training set consists of 140 generated instances following Taillard-style processing-time distributions. The test set contains standard DMU benchmarks. The structural objective is discovered using a 100k training budget and evaluated under both 100k and 400k iteration budgets. The original objective is the makespan as shown in Eq. 42. JOR modifies only the move evaluation score used by tabu search (Li et al., 2026). The final schedule is evaluated using the original makespan objective.

LLM Inference Acceleration. We evaluate JOR on speculative decoding using EAGLE-3 (Li et al., 2025b). The target model is Vicuna-13B-v1.3 with the official EAGLE-3 draft model. Decoding follows the official EAGLE implementation with: $\texttt{total_token} = 60$, $\texttt{depth} = 7$, $\texttt{top_k} = 10$. The stock EAGLE-3 configuration expands the top-k beam nodes for a fixed depth of seven layers, with cumulative draft-model log probability used for node ranking. JOR leaves this ranking score unchanged and reshapes only the expansion gate defined in Eq. 46, allowing the verification tree to adapt its expansion pattern while preserving the original ranking and verification procedure.

The number of datesets are: $N_{\text{inner}} = 120$, $N_{\text{gate}} = 120$, $N_{\text{test}} = 480$. Final evaluation additionally follows the official EAGLE benchmark protocol with six tasks and 480 prompts. The 120 inner prompts and 120 gate prompts used during objective discovery are disjoint from each other and from the separate 480-prompt Spec-Bench evaluation set. The gate prompts constitute $\mathcal{D}_{\text{align}}$ for the alignment acceptance criterion; verifier-derived reference information on these prompts is used only by that gate. No evaluation prompt is used during structural search, numerical optimization, or gating. All experiments use greedy decoding ($T = 0$) with fixed target and draft models.

Bayesian Optimization. We evaluate JOR on black-box Bayesian optimization following FunBO (Aglietti et al., 2024)-style protocols. The baseline optimizer uses Gaussian process regression with fixed RBF kernel hyperparameters and expected improvement (EI) acquisition. The baseline acquisition rule is Eq. 50. And JOR modifies only the acquisition score in Eq. 51.

The objective functions include both in-distribution and out-of-distribution benchmark functions, including Ackley, Levy, Rosenbrock, Branin, Goldstein–Price, Hartmann, and additional synthetic black-box functions. And the benchmark contains four protocols: (1) OOD benchmark, (2) ID benchmark, (3) few-shot adaptation, (4) AdaBoost hyperparameter optimization.

For synthetic functions, the optimization budget is $T = 30$ queries, with additional evaluations at $T = 20$ and $T = 50$. Performance is measured by normalized optimization cost and convergence speed. The true objective function is never provided to the structural proposer or zeroth-order optimization during shaping; on $\mathcal{D}_{\text{align}}$, offline objective values are used only to construct the reference region for the alignment acceptance criterion.

Reinforcement Learning Reward Shaping. We evaluate JOR on sparse reward reinforcement learning using CartPole, following reward shaping studies such as BiPaRS (Hu et al., 2020). The environment reward is shown in Eq. 56. The policy optimizer is PPO with a fixed architecture and training procedure. The policy network uses two hidden layers, and updates are performed with a fixed number of environment interactions. JOR modifies only the training reward Eq. 57.

Experiments contain 15 environment variants with different physical parameters, including pole length, masses, gravity, and force magnitude. The dataset contains 15×100 environment seeds. And training uses eight environment variants, while seven unseen variants are reserved for final evaluation. The PPO training budgets include: $100k$, $200k$, $400k$, $1.2M$ environment steps. Performance is evaluated using the original environment reward through average success performance (ASPE). The shaped reward is never used for evaluation.

Objective-Discovery Computational Cost. To quantify the computational cost of objective discovery, we report the input, output, and total LLM token consumption, together with the end-to-end wall-clock discovery time, for JOR and each LLM-based baseline Table 6. Within each domain, all methods are given roughly comparable wall-clock budgets for objective discovery, rather than being matched by token consumption. The reported discovery time covers the complete procedure required to obtain the final guidance objective, including one-time construction of the reference

Table 6: Computational cost of objective discovery across domains and methods.

Domain	Method	Input Tokens	Output Tokens	Total Tokens	Discovery Time
CVRP	Claude-goal	48.9M	0.38M	49.3M	7.9 h
	ShinkaEvolve	6.8M	1.38M	8.2M	8.2 h
	CORAL	60M	0.45M	60.5M	8.0 h
	JOR	2.5M	2.2M	4.7M	8.0 h
EAGLE	Claude-goal	60.4M	0.3M	60.7M	8.3 h
	ShinkaEvolve	3.7M	0.37M	4.1M	8.0 h
	CORAL	1413M	1.9M	1415M	8.7 h
	JOR	1.7M	1.4M	3.1M	8.3 h
BO	Claude-goal	46M	0.3M	46.3M	4.0 h
	ShinkaEvolve	3.8M	0.8M	4.6M	4.5 h
	CORAL	480M	0.6M	480.6M	4.3 h
	JOR	1.8M	1.5M	3.3M	4.1 h
RL	Claude-goal	14M	0.12M	14.12M	8.0 h
	ShinkaEvolve	5.7M	1.2M	6.9M	8.2 h
	CORAL	810M	1.2M	811.2M	8.5 h
	JOR	3.1M	3.0M	6.1M	7.9 h

region which is done in parallel, LLM interaction, candidate evaluation, and method-specific optimization or evolution, while excluding final test-set evaluation. Because different methods employ substantially different prompting, memory, and tool-interaction strategies, their token consumption can vary considerably even under similar wall-clock budgets. All measurements are collected under the same hardware and software environment whenever applicable.

D.2 Discovered Guidance Objectives

To make the structural objective discovery in JOR fully transparent, we report the objection functions obtained in all experimental domains. For a frozen Bank $\mathcal{K}^\star$, the discovered guidance objective is instantiated as

$$G^\star(v;x) = G_0(v;x) + \sum_{k\in\mathcal{K}^\star} \lambda_k \phi_k(v;x;\theta_k). \tag{60}$$

The following tables report all accepted shaping components together with their coefficients and numerical parameters. The reported checkpoints are the ones used for the corresponding experimental evaluations unless otherwise noted.

LKH-CVRP.

$$\begin{aligned} G = G_0 &+ 0.0149\,\phi_{\text{periodic-capacity-fill}} + 1.2180\,\phi_{\text{angular-rank-joint}} \\ &+ 0.1000\,\phi_{\text{flexible-angular-rank}} + 0.3000\,\phi_{\text{adaptive-scale-capacity-reach}} \\ &+ 0.3000\,\phi_{\text{route-size-scaled-reach}} + 0.3000\,\phi_{\text{savings-gated-rank-reach}} \\ &+ 0.1000\,\phi_{\text{burden-weighted-local-scale}}. \end{aligned}$$

LKH Functions

```
# ---- periodic_capacity_fill_bonus  (lambda = 0.0149)
def periodic_capacity_fill_bonus(ctx):
    A = 0.513; sigma = 0.471; max_mult = 2.917
    demand = ctx.demand; capacity = ctx.capacity
    frac = demand / (capacity + 1e-9)
    s = frac[:, None] + frac[None, :]
    mult = np.clip(np.round(s), 1.0, max_mult)
    dev = np.abs(s - mult)
    out = -A * np.exp(-(dev ** 2) / (2 * sigma ** 2 + 1e-9))
    np.fill_diagonal(out, 0.0)
    return out

# ---- angular_rank_joint_penalty  (lambda = 1.218)
def angular_rank_joint_penalty(ctx):
    A = 0.215; f = 0.131; p = 2.987
    x = ctx.x; y = ctx.y; dist = ctx.dist; is_depot = ctx.is_depot
```

```
    n = dist.shape[0]
    dx = x - x[0]; dy = y - y[0]
    theta = np.arctan2(dy, dx)
    adiff = np.abs(theta[:, None] - theta[None, :])
    adiff = np.minimum(adiff, 2 * np.pi - adiff) / np.pi
    order = np.argsort(dist, axis=1)
    rank = np.empty_like(order)
    rows = np.arange(n)[:, None]
    rank[rows, order] = np.arange(n)[None, :]
    rank_frac = rank / (n - 1)
    rdiff = np.maximum(rank_frac, rank_frac.T)
    rdiff = np.clip(rdiff - f, 0.0, None) / (1.0 - f + 1e-9)
    out = A * (adiff * rdiff) ** p
    depot_mask = is_depot[:, None] | is_depot[None, :]
    out = np.where(depot_mask, 0.0, out)
    np.fill_diagonal(out, 0.0)
    return out

# ---- flexible_angular_rank_penalty  (lambda = 0.1)
def flexible_angular_rank_penalty(ctx):
    A = 1.323; f_r = 0.123; f_a = 0.07; p_a = 1.793; p_r = 1.979; blend
     = 0.943
    x = ctx.x; y = ctx.y; dist = ctx.dist; is_depot = ctx.is_depot
    n = dist.shape[0]
    dx = x - x[0]; dy = y - y[0]
    theta = np.arctan2(dy, dx)
    adiff = np.abs(theta[:, None] - theta[None, :])
    adiff = np.minimum(adiff, 2 * np.pi - adiff) / np.pi
    adiff = np.clip(adiff - f_a, 0.0, None) / (1.0 - f_a + 1e-9)
    order = np.argsort(dist, axis=1)
    rank = np.empty_like(order)
    rows = np.arange(n)[:, None]
    rank[rows, order] = np.arange(n)[None, :]
    rank_frac = rank / (n - 1 + 1e-9)
    rdiff = np.maximum(rank_frac, rank_frac.T)
    rdiff = np.clip(rdiff - f_r, 0.0, None) / (1.0 - f_r + 1e-9)
    prod_term = (adiff ** p_a) * (rdiff ** p_r)
    mean_term = 0.5 * (adiff ** p_a + rdiff ** p_r)
    out = A * (blend * prod_term + (1.0 - blend) * mean_term)
    depot_mask = is_depot[:, None] | is_depot[None, :]
    out = np.where(depot_mask, 0.0, out)
    np.fill_diagonal(out, 0.0)
    return out

# ---- adaptive_scale_capacity_reach_penalty  (lambda = 0.3)
def adaptive_scale_capacity_reach_penalty(ctx):
    A = 0.5; f = 0.1; reach0 = 2.0; p = 1.5; alpha_r = 1.0; alpha_l =
    0.0
    dist = ctx.dist; demand = ctx.demand; capacity = ctx.capacity;
    is_depot = ctx.is_depot
    n = dist.shape[0]
    k = max(1, int(f * n))
    order = np.argsort(dist, axis=1)
    neighbor_idx = order[:, 1:k + 1]
    local_demand = np.sum(demand[neighbor_idx], axis=1) + demand
    L = local_demand / (capacity + 1e-9)
    looseness = 1.0 / (1.0 + L)
    rows = np.arange(n)[:, None]
    Rk = dist[rows, neighbor_idx][:, -1]
    Rk = np.maximum(Rk, 1e-6)
    reach_i = dist / Rk[:, None]
    reach_j = dist / Rk[None, :]
    reach_min = np.minimum(reach_i, reach_j)
    reach_max = np.maximum(reach_i, reach_j)
    reach = alpha_r * reach_min + (1.0 - alpha_r) * reach_max
```

```
    excess = np.clip(reach - reach0, 0.0, None)
    loose_min = np.minimum(looseness[:, None], looseness[None, :])
    loose_max = np.maximum(looseness[:, None], looseness[None, :])
    looseness_pair = alpha_l * loose_min + (1.0 - alpha_l) * loose_max
    out = A * (excess ** p) * looseness_pair
    depot_mask = is_depot[:, None] | is_depot[None, :]
    out = np.where(depot_mask, 0.0, out)
    np.fill_diagonal(out, 0.0)
    return out

# ---- route_size_scaled_reach_penalty  (lambda = 0.3)
def route_size_scaled_reach_penalty(ctx):
    A = 0.18; f = 1.452; reach0 = 2.644; p = 2.775
    dist = ctx.dist; demand = ctx.demand; capacity = ctx.capacity;
    is_depot = ctx.is_depot
    n = dist.shape[0]
    customer_demand = demand[~is_depot]
    mean_demand = np.mean(customer_demand) + 1e-9
    r_est = capacity / mean_demand
    k = int(np.clip(round(f * r_est), 1, n - 1))
    order = np.argsort(dist, axis=1)
    neighbor_idx = order[:, 1:k + 1]
    rows = np.arange(n)[:, None]
    local_demand = np.sum(demand[neighbor_idx], axis=1) + demand
    L = local_demand / (capacity + 1e-9)
    looseness = 1.0 / (1.0 + L)
    Rk = dist[rows, neighbor_idx][:, -1]
    Rk = np.maximum(Rk, 1e-6)
    reach_i = dist / Rk[:, None]
    reach_j = dist / Rk[None, :]
    reach = np.minimum(reach_i, reach_j)
    excess = np.clip(reach - reach0, 0.0, None)
    loose_pair = np.minimum(looseness[:, None], looseness[None, :])
    out = A * (excess ** p) * loose_pair
    depot_mask = is_depot[:, None] | is_depot[None, :]
    out = np.where(depot_mask, 0.0, out)
    np.fill_diagonal(out, 0.0)
    return out

# ---- savings_gated_rank_reach_penalty  (lambda = 0.3)
def savings_gated_rank_reach_penalty(ctx):
    A = 1.346; f = 0.065; rank0 = 0.581; p = 1.681; tau_s = 1.0
    dist = ctx.dist; demand = ctx.demand; capacity = ctx.capacity;
    is_depot = ctx.is_depot
    n = dist.shape[0]
    k = max(1, int(f * n))
    order = np.argsort(dist, axis=1)
    neighbor_idx = order[:, 1:k + 1]
    rows = np.arange(n)[:, None]
    local_demand = np.sum(demand[neighbor_idx], axis=1) + demand
    L = local_demand / (capacity + 1e-9)
    looseness = 1.0 / (1.0 + L)
    rank = np.empty_like(order)
    rank[rows, order] = np.arange(n)[None, :]
    rank_frac = rank / (n - 1 + 1e-9)
    reach = np.minimum(rank_frac, rank_frac.T)
    excess = np.clip(reach - rank0, 0.0, None) / (1.0 - rank0 + 1e-9)
    loose_pair = np.minimum(looseness[:, None], looseness[None, :])
    depot_dist = dist[0, :]
    di = depot_dist[:, None]; dj = depot_dist[None, :]
    savings = di + dj - dist
    denom = di + dj + 1e-9
    sratio = savings / denom
    gate = 1.0 - np.clip((sratio - tau_s) / (1.0 - tau_s + 1e-9), 0.0,
    1.0)
```

```
    out = A * (excess ** p) * loose_pair * gate
    depot_mask = is_depot[:, None] | is_depot[None, :]
    out = np.where(depot_mask, 0.0, out)
    np.fill_diagonal(out, 0.0)
    return out

# ---- burden_weighted_local_scale_penalty  (lambda = 0.1)
def burden_weighted_local_scale_penalty(ctx):
    A = 0.247; f = 1.55; reach0 = 1.974; p = 1.038; e_d = 1.12; e_r =
    0.977
    x = ctx.x; y = ctx.y; dist = ctx.dist; demand = ctx.demand;
    capacity = ctx.capacity; is_depot = ctx.is_depot
    n = dist.shape[0]
    customer_demand = demand[~is_depot]
    mean_demand = np.mean(customer_demand) + 1e-9
    r_est = capacity / mean_demand
    k = int(np.clip(np.round(f * r_est), 1, n - 1))
    order = np.argsort(dist, axis=1)
    neighbor_idx = order[:, 1:k + 1]
    rows = np.arange(n)[:, None]
    Rk = dist[rows, neighbor_idx][:, -1]
    Rk = np.maximum(Rk, 1e-6)
    reach_i = dist / Rk[:, None]; reach_j = dist / Rk[None, :]
    reach = np.minimum(reach_i, reach_j)
    excess = np.clip(reach - reach0, 0.0, None)
    dx = x - x[0]; dy = y - y[0]
    r = np.sqrt(dx ** 2 + dy ** 2)
    mean_r = np.mean(r[~is_depot]) + 1e-9
    burden = (demand / (capacity + 1e-9)) ** e_d * (r / mean_r) ** e_r
    burden_pair = np.minimum(burden[:, None], burden[None, :])
    out = A * (excess ** p) * burden_pair
    depot_mask = is_depot[:, None] | is_depot[None, :]
    out = np.where(depot_mask, 0.0, out)
    np.fill_diagonal(out, 0.0)
    return out
```

PyVRP-CVRP.

$$G = G_0 - 0.0300\,\phi_{\text{angular-sector-alignment}} + 0.0300\,\phi_{\text{savings-arc-attraction}} - 0.0300\,\phi_{\text{isolated-node-lock-in}} + 0.3000\,\phi_{\text{bhh-area-route-scale}} - 0.0300\,\phi_{\text{demand-variance-scale-adaptive}}.$$

PyVRP Functions

```
# ---- angular_sector_alignment  (lambda = -0.03)
def angular_sector_alignment(ctx):
    A = 0.362; p = 1.029; wcap = 3.878
    x = ctx.x; y = ctx.y; dist = ctx.dist
    dx = x - x[0]; dy = y - y[0]
    theta = np.arctan2(dy, dx)
    diff = np.abs(theta[:, None] - theta[None, :])
    angdiff = np.minimum(diff, 2 * np.pi - diff) / np.pi
    d0 = dist[0]
    denom = d0.mean() + 1e-9
    w = np.minimum(d0 / denom, wcap)
    wij = np.minimum(w[:, None], w[None, :])
    out = A * (angdiff ** p) * wij
    return out

# ---- savings_arc_attraction  (lambda = 0.03)
def savings_arc_attraction(ctx):
    A = 0.964; tau = 0.793; p = 1.709
    dist = ctx.dist
```

```
    eps = 1e-9
    d0 = dist[0]
    savings = d0[:, None] + d0[None, :] - dist
    denom = d0[:, None] + d0[None, :] + eps
    r = savings / denom
    diff = tau - r
    out = A * np.sign(diff) * np.abs(diff) ** p
    return out

# ---- isolated_node_lock_in_bias  (lambda = -0.03)
def isolated_node_lock_in_bias(ctx):
    A = 0.901; tau = 2.327; p = 0.897
    dist = ctx.dist; is_depot = ctx.is_depot
    eps = 1e-9
    dd = dist.copy()
    np.fill_diagonal(dd, np.inf)
    dd[:, is_depot] = np.inf
    dd[is_depot, :] = np.inf
    rowmin = dd.min(axis=1)
    med = np.median(rowmin[np.isfinite(rowmin)]) + eps
    rowmin = np.where(np.isfinite(rowmin), rowmin, med)
    isolation = rowmin / med
    ratio_i = dist / (rowmin[:, None] + eps)
    ratio_j = dist / (rowmin[None, :] + eps)
    ratio = np.minimum(ratio_i, ratio_j)
    weight = np.maximum(isolation[:, None], isolation[None, :])
    diff = ratio - tau
    out = A * weight * np.sign(diff) * np.abs(diff) ** p
    return out

# ---- bhh_area_route_scale_bias  (lambda = 0.3)
def bhh_area_route_scale_bias(ctx):
    A = 0.5; tau = 1.0; p = 1.0
    x = ctx.x; y = ctx.y; dist = ctx.dist; demand = ctx.demand;
    capacity = ctx.capacity; is_depot = ctx.is_depot
    eps = 1e-9
    cust = ~is_depot
    xc = x[cust]; yc = y[cust]
    area = (xc.max() - xc.min() + eps) * (yc.max() - yc.min() + eps)
    k_est = max(demand.sum() / (capacity + eps), 1.0)
    scale = np.sqrt(area / k_est) + eps
    ratio = dist / scale
    diffv = ratio - tau
    out = A * np.sign(diffv) * np.abs(diffv) ** p
    return out

# ---- demand_variance_scale_adaptive_bias  (lambda = -0.03)
def demand_variance_scale_adaptive_bias(ctx):
    A = 1.607; tau = 2.269; p = 0.556; kappa = 1.661
    dist = ctx.dist; demand = ctx.demand; is_depot = ctx.is_depot
    eps = 1e-9
    cust = ~is_depot
    dc = demand[cust]
    mean_d = dc.mean() + eps
    cv = dc.std() / mean_d
    base_scale = dist[np.ix_(cust, cust)].mean() + eps
    adjusted = base_scale * (1.0 + kappa * cv) + eps
    ratio = dist / adjusted
    diff = ratio - tau
    out = A * np.sign(diff) * np.abs(diff) ** p
    return out
```

JSSP.

$$G = G_0 + 0.3000\,\phi_{\text{job-completion-fraction}} + 0.1000\,\phi_{\text{machine-local-tail-rank}} + 0.0300\,\phi_{\text{machine-congestion-head-fairness}}.$$

JSSP Functions

```
# ---- job_completion_fraction_bias  (lambda = 0.3)
def job_completion_fraction_bias(ctx):
    scale = 0.5; power = 1.0
    job_head = ctx.job_head; job_total = ctx.job_total
    frac_complete = job_head / (job_total + 1e-9)
    diff = frac_complete[:, None] - frac_complete[None, :]
    x = np.tanh(diff / (scale + 1e-9))
    out = np.sign(x) * np.abs(x) ** power
    return out

# ---- machine_local_tail_rank_bias  (lambda = 0.1)
def machine_local_tail_rank_bias(ctx):
    power = 1.0; temp = 0.5
    job_tail = ctx.job_tail
    same_machine = ctx.same_machine
    less = (job_tail[None, :] < job_tail[:, None]) & same_machine
    cnt = same_machine.sum(axis=1)
    rank = less.sum(axis=1) / (cnt + 1e-9)
    diff = rank[None, :] - rank[:, None]
    x = np.tanh(diff / (temp + 1e-9))
    out = np.sign(x) * np.abs(x) ** power
    return out

# ---- machine_congestion_head_fairness_gate  (lambda = 0.03)
def machine_congestion_head_fairness_gate(ctx):
    K = 2.653; rank_tau = 0.306; power = 1.36
    mach_load = ctx.mach_load; job_head = ctx.job_head; p_mean = ctx.
    p_mean
    load_rank = (mach_load[None, :] < mach_load[:, None]).mean(axis=1)
    crit = np.clip((load_rank - rank_tau) / (1.0 - rank_tau + 1e-9),
    0.0, 1.0)
    gate = 0.5 * (crit[:, None] + crit[None, :])
    diff = job_head[:, None] - job_head[None, :]
    sig = np.tanh(diff / (p_mean + 1e-9))
    out = K * gate * np.sign(sig) * np.abs(sig) ** power
    return out
```

LLM inference acceleration.

$$G = G_0 + 0.1641\,\phi_{\text{sibling-root-deficit-ratio}} + 0.9981\,\phi_{\text{family-best-case-root-margin}} - 0.2133\,\phi_{\text{rank-weighted-layer-frontier-margin}} - 5.6451\,\phi_{\text{layer-pool-root-drift-margin}} + 4.0000\,\phi_{\text{root-deficit-scaled-by-entropy}}.$$

LLM Inference Acceleration Functions

```
# Expansion decision at draft layer L: expand beam node v iff G(v) >=
    0, where
# G_0(v) = +1 for L <= 7 and -1 for deeper layers (the stock depth-7
    prior), and every
# phi below is normalised per (prompt row, layer) to mean |phi| = 1
    before weighting.
# Verification keeps the stock rule: prefix-closed top-60 by cum_logp
    over drafted nodes.

# ---- sibling_root_deficit_ratio  (lambda = 0.1641)
def sibling_root_deficit_ratio(ctx):
```

```
    A = 1.376; tau = 0.872
    cum_logp = ctx.cum_logp; sib_max_cum = ctx.sib_max_cum;
    root_max_cum = ctx.root_max_cum
    eps = 1e-6
    deficit_sib = sib_max_cum - cum_logp
    deficit_root = root_max_cum - cum_logp
    ratio = deficit_sib / (deficit_root + eps)
    return np.tanh(A * (tau - ratio))

# ---- family_best_case_root_margin  (lambda = 0.9981)
def family_best_case_root_margin(ctx):
    A = 1.278; tau = 0.574
    root_max_cum = ctx.root_max_cum; sib_max_cum = ctx.sib_max_cum;
    base_spread = ctx.base_spread
    eps = 1e-6
    best_case_deficit = (root_max_cum - sib_max_cum) / (base_spread[:,
    None] + eps)
    return np.tanh(A * (tau - best_case_deficit))

# ---- rank_weighted_layer_frontier_margin  (lambda = -0.2133)
def rank_weighted_layer_frontier_margin(ctx):
    A = 3.582; K = 1.253
    cum_logp = ctx.cum_logp; layer_max_cum = ctx.layer_max_cum
    root_max_cum = ctx.root_max_cum; parent_cum = ctx.parent_cum;
    sib_rank = ctx.sib_rank
    eps = 1e-6
    scale = (root_max_cum - parent_cum) + eps
    diff = (cum_logp - layer_max_cum) / scale
    weight = 1.0 / (1.0 + K * sib_rank.astype(np.float64))
    return np.tanh(A * diff * weight)

# ---- layer_pool_root_drift_margin  (lambda = -5.6451)
def layer_pool_root_drift_margin(ctx):
    A = 2.219; tau = 1.798
    root_max_cum = ctx.root_max_cum; layer_beam_mean_cum = ctx.
    layer_beam_mean_cum
    layer_max_cum = ctx.layer_max_cum
    eps = 1e-6
    scale = (root_max_cum - layer_max_cum) + eps
    drift = (root_max_cum - layer_beam_mean_cum) / scale
    return np.tanh(A * (tau - drift))

# ---- root_deficit_scaled_by_entropy  (lambda = 4.0000)
def root_deficit_scaled_by_entropy(ctx):
    A = 2.999; tau = 3.292
    root_max_cum = ctx.root_max_cum; cum_logp = ctx.cum_logp; entropy =
     ctx.entropy
    eps = 1e-6
    ratio = (root_max_cum - cum_logp) / (entropy + eps)
    return np.tanh(A * (tau - ratio))
```

Bayesian optimization.

$$
\begin{aligned}
G = \mathrm{EI} &+ 0.4755\,\phi_{\text{boundary-affinity}} + 0.2731\,\phi_{\text{boundary-topk-saturating-affinity}} \\
&+ 0.3363\,\phi_{\text{boundary-density-interaction}} - 0.0674\,\phi_{\text{local-envelope-overshoot}} \\
&+ 0.4174\,\phi_{\text{local-knn-idw-disagreement}} - 0.3000\,\phi_{\text{envelope-overshoot-asymmetric}} \\
&+ 0.3000\,\phi_{\text{axis-novelty-landscape-gated}} + 0.1000\,\phi_{\text{boundary-value-outlier-interaction}} \\
&- 0.0300\,\phi_{\text{ei-decomposition-exploit-bias}} + 0.1000\,\phi_{\text{linear-trend-extrapolation}} \\
&+ 0.3000\,\phi_{\text{quantile-gap-confidence-asymmetric}}.
\end{aligned}
$$

Bayesian Optimization Functions

```
# next query = argmax over unobserved grid points of EI(x) + sum_k
    lambda_k * phi_k(ctx);
# every phi_k is centred over the candidates and rescaled to the EI
    spread of the step
# (ctx.base_spread) before weighting. Row arrays are shaped (1, n_cands
    ).

# ---- boundary_affinity  (lambda = 0.4755)
def boundary_affinity(ctx):
    power = 2.341
    x = ctx.x
    edge = np.minimum(x, 1.0 - x).min(axis=1)
    out = edge ** power
    return out.reshape(1, -1)

# ---- boundary_topk_saturating_affinity  (lambda = 0.2731)
def boundary_topk_saturating_affinity(ctx):
    power = 2.519; dim_frac = 0.873; cap = 0.335
    x = ctx.x
    edge = np.minimum(x, 1.0 - x)
    d = edge.shape[1]
    k = max(1, int(np.ceil(dim_frac * d)))
    agg = np.sort(edge, axis=1)[:, :k].mean(axis=1)
    out = (np.minimum(agg, cap) / cap) ** power
    return out.reshape(1, -1)

# ---- boundary_density_interaction  (lambda = 0.3363)
def boundary_density_interaction(ctx):
    power_edge = 0.439; power_dist = 2.527; eps = 0.042
    x = ctx.x; dist_obs = ctx.dist_obs.reshape(-1)
    edge = np.minimum(x, 1.0 - x).min(axis=1)
    out = (edge + eps) ** power_edge * (dist_obs + eps) ** power_dist
    return out.reshape(1, -1)

# ---- local_envelope_overshoot  (lambda = -0.0674)
def local_envelope_overshoot(ctx):
    k_frac = 0.500; eps = 0.0035
    x = ctx.x; x_obs = ctx.x_obs; y_obs = ctx.y_obs
    mean = ctx.mean.reshape(-1); obs_spread = ctx.obs_spread
    n_cands = x.shape[0]; n_obs = x_obs.shape[0]
    if n_obs < 2:
        return np.zeros((1, n_cands))
    dist = np.sqrt(np.sum((x[:, None, :] - x_obs[None, :, :]) ** 2,
    axis=2) + eps ** 2)
    k = min(max(2, int(np.ceil(k_frac * n_obs))), n_obs)
    neigh_y = y_obs[np.argsort(dist, axis=1)[:, :k]]
    over_high = np.maximum(mean - neigh_y.max(axis=1), 0.0)
    over_low = np.maximum(neigh_y.min(axis=1) - mean, 0.0)
    out = (over_high - over_low) / (obs_spread + eps)
    return out.reshape(1, -1)

# ---- local_knn_idw_disagreement  (lambda = 0.4174)
def local_knn_idw_disagreement(ctx):
    p = 1.618; eps = 0.0069; k_frac = 0.593
    x = ctx.x; x_obs = ctx.x_obs; y_obs = ctx.y_obs
    mean = ctx.mean.reshape(-1); obs_spread = ctx.obs_spread
    n_cands = x.shape[0]; n_obs = x_obs.shape[0]
    if n_obs < 2:
        return np.zeros((1, n_cands))
    dist = np.sqrt(np.sum((x[:, None, :] - x_obs[None, :, :]) ** 2,
    axis=2) + eps ** 2)
    k = min(max(2, int(np.ceil(k_frac * n_obs))), n_obs)
    order = np.argsort(dist, axis=1)[:, :k]
    w = 1.0 / (np.take_along_axis(dist, order, axis=1) ** p)
```

```
    idw = (w * y_obs[order]).sum(axis=1) / w.sum(axis=1)
    out = (mean - idw) / (obs_spread + eps)
    return out.reshape(1, -1)

# ---- envelope_overshoot_asymmetric  (lambda = -0.3)
def envelope_overshoot_asymmetric(ctx):
    k_frac = 0.578; eps = 0.036; w_high = 0.281; w_low = 1.566;
    std_blend = 0.676
    x = ctx.x; x_obs = ctx.x_obs; y_obs = ctx.y_obs
    mean = ctx.mean.reshape(-1); std = ctx.std.reshape(-1); obs_spread
    = ctx.obs_spread
    n_cands = x.shape[0]; n_obs = x_obs.shape[0]
    if n_obs < 2:
        return np.zeros((1, n_cands))
    dist = np.sqrt(np.sum((x[:, None, :] - x_obs[None, :, :]) ** 2,
    axis=2) + eps ** 2)
    k = min(max(2, int(np.ceil(k_frac * n_obs))), n_obs)
    neigh_y = y_obs[np.argsort(dist, axis=1)[:, :k]]
    denom = (1.0 - std_blend) * (obs_spread + eps) + std_blend * (std +
     eps)
    over_high = np.maximum(mean - neigh_y.max(axis=1), 0.0)
    over_low = np.maximum(neigh_y.min(axis=1) - mean, 0.0)
    out = (w_high * over_high - w_low * over_low) / denom
    return out.reshape(1, -1)

# ---- axis_novelty_landscape_gated  (lambda = 0.3)
def axis_novelty_landscape_gated(ctx):
    frac = 0.905; eps = 1.43e-4; power = 2.382; gate_scale = 4.490;
    gate_power = 1.723
    x = ctx.x; x_obs = ctx.x_obs; y_obs = ctx.y_obs
    n_cands, d = x.shape; n_obs = x_obs.shape[0]
    if n_obs < 1:
        return np.zeros((1, n_cands))
    x_inc = x_obs[np.argmin(y_obs)]
    gap = np.abs(x[:, None, :] - x_obs[None, :, :]).min(axis=1)
    dist_inc_axis = np.abs(x - x_inc[None, :])
    ratio = gap / (gap + dist_inc_axis + eps)
    k = max(1, int(np.ceil(frac * d)))
    agg = np.sort(ratio, axis=1)[:, ::-1][:, :k].mean(axis=1)
    base_term = agg ** power
    base_spread = float(np.ravel(ctx.base_spread)[0]); obs_spread =
    float(np.ravel(ctx.obs_spread)[0])
    s = (base_spread / (obs_spread + eps)) * gate_scale
    gate = s ** gate_power / (1.0 + s ** gate_power)
    out = base_term * gate
    return out.reshape(1, -1)

# ---- boundary_value_outlier_interaction  (lambda = 0.1)
def boundary_value_outlier_interaction(ctx):
    power_edge = 0.499; power_gap = 0.821; center = 0.671; eps = 3.29e
    -4
    x = ctx.x; mean = ctx.mean.reshape(-1); y_obs = ctx.y_obs
    n_cands = x.shape[0]; n_obs = y_obs.shape[0]
    edge = np.minimum(x, 1.0 - x).min(axis=1)
    if n_obs < 1:
        return np.zeros((1, n_cands))
    frac_below = (y_obs[None, :] < mean[:, None]).mean(axis=1)
    gap = np.abs(frac_below - center)
    out = (edge + eps) ** power_edge * gap ** power_gap
    return out.reshape(1, -1)

# ---- ei_decomposition_exploit_bias  (lambda = -0.03)
def ei_decomposition_exploit_bias(ctx):
    center = 0.238; power = 2.144; eps = 0.0013
```

```
    z = ctx.z.reshape(-1); cdf = ctx.cdf.reshape(-1); pdf = ctx.pdf.
    reshape(-1)
    exploit = z * cdf
    exploit_frac = exploit / (np.abs(exploit) + pdf + eps)
    diff = exploit_frac - center
    out = np.sign(diff) * np.abs(diff) ** power
    return out.reshape(1, -1)

# ---- linear_trend_extrapolation  (lambda = 0.1)
def linear_trend_extrapolation(ctx):
    ridge = 0.051; decay = 1.426; power = 0.891
    x = ctx.x; x_obs = ctx.x_obs; y_obs = ctx.y_obs
    n_cands, d = x.shape; n_obs = x_obs.shape[0]
    if n_obs < 2:
        return np.zeros((1, n_cands))
    w = np.exp(-decay * (n_obs - 1 - np.arange(n_obs)))
    Xd = np.hstack([x_obs, np.ones((n_obs, 1))])
    coef = np.linalg.solve(Xd.T @ (w[:, None] * Xd) + ridge * np.eye(d
    + 1), Xd.T @ (w * y_obs))
    pred = x @ coef[:-1] + coef[-1]
    zscore = -(pred - pred.mean()) / (pred.std() + 1e-8)
    out = np.sign(zscore) * np.abs(zscore) ** power
    return out.reshape(1, -1)

# ---- quantile_gap_confidence_asymmetric  (lambda = 0.3)
def quantile_gap_confidence_asymmetric(ctx):
    power = 0.306; center = 0.226; w_low = 0.160; w_high = 0.017;
    conf_frac = 0.547; gate_power = 1.969
    mean = ctx.mean.reshape(-1); y_obs = ctx.y_obs; T = ctx.T
    n_cands = mean.shape[0]; n_obs = y_obs.shape[0]
    if n_obs < 1:
        return np.zeros((1, n_cands))
    diff = (y_obs[None, :] < mean[:, None]).mean(axis=1) - center
    pos = np.maximum(diff, 0.0) ** power
    neg = np.maximum(-diff, 0.0) ** power
    gate = min(1.0, n_obs / max(conf_frac * T, 1e-8)) ** gate_power
    out = gate * (w_low * neg - w_high * pos)
    return out.reshape(1, -1)
```

Reinforcement learning.

$$G = r + 1.0000\,\phi_{\text{recovery-crossing-bonus}} + 0.3000\,\phi_{\text{joint-axis-safety}} + 0.3000\,\phi_{\text{angular-recovery-velocity}} - 0.3000\,\phi_{\text{cart-time-to-boundary}} - 0.3000\,\phi_{\text{kinematic-time-to-impact}}.$$

Reinforcement Learning Functions

```
# training reward = r + sum_k lambda_k * phi_k(ctx) / s_k, where r is
    the sparse true reward
# (-1 on failure, 0 otherwise) and s_k is the mean |phi_k| on a fixed
    probe buffer;
# one column per transition; x_thresh = 2.4, theta_thresh = 12 deg,
    gamma = 0.999.

# ---- recovery_crossing_bonus  (lambda = 1.0)
def recovery_crossing_bonus(ctx):
    A = 2.357; cap = 0.489
    theta = ctx.theta; theta_next = ctx.theta_next; theta_thresh = ctx.
    theta_thresh
    crossed = (np.sign(theta) * np.sign(theta_next) < 0.0).astype(np.
    float32)
    severity = np.clip(np.abs(theta) / theta_thresh, 0.0, cap)
    out = A * crossed * severity
```

```
    return out

# ---- joint_axis_safety  (lambda = 0.3)
def joint_axis_safety(ctx):
    A = 1.0; p = 1.0
    x = ctx.x; x_next = ctx.x_next; theta = ctx.theta; theta_next = ctx
    .theta_next
    x_thresh = ctx.x_thresh; theta_thresh = ctx.theta_thresh; gamma =
    ctx.gamma
    mx = 1.0 - np.clip(np.abs(x) / x_thresh, 0.0, 1.0)
    mth = 1.0 - np.clip(np.abs(theta) / theta_thresh, 0.0, 1.0)
    mx_next = 1.0 - np.clip(np.abs(x_next) / x_thresh, 0.0, 1.0)
    mth_next = 1.0 - np.clip(np.abs(theta_next) / theta_thresh, 0.0,
    1.0)
    Phi = (mx ** p) * (mth ** p)
    Phi_next = (mx_next ** p) * (mth_next ** p)
    out = A * (gamma * Phi_next - Phi)
    return out

# ---- angular_recovery_velocity  (lambda = 0.3)
def angular_recovery_velocity(ctx):
    A = 0.227; tau = 0.800
    theta = ctx.theta; theta_dot = ctx.theta_dot; theta_thresh = ctx.
    theta_thresh
    moving_toward_center = -np.tanh((theta * theta_dot) / (theta_thresh
     * tau))
    severity = np.clip(np.abs(theta) / theta_thresh, 0.0, 1.0)
    out = A * moving_toward_center * severity
    return out

# ---- cart_time_to_boundary  (lambda = -0.3)
def cart_time_to_boundary(ctx):
    A = 2.236; tau = 1.918
    x = ctx.x; x_dot = ctx.x_dot; x_thresh = ctx.x_thresh
    dist = np.clip(x_thresh - np.abs(x), 0.0, x_thresh)
    approach = np.sign(x) * x_dot
    danger = approach * x_thresh / (dist + 0.05 * x_thresh)
    out = -A * np.tanh(danger * tau)
    return out

# ---- kinematic_time_to_impact  (lambda = -0.3)
def kinematic_time_to_impact(ctx):
    A = 1.0; tau = 1.0
    x = ctx.x; x_dot = ctx.x_dot; x_thresh = ctx.x_thresh
    masscart = ctx.masscart; force_mag = ctx.force_mag
    remaining = np.clip(x_thresh - np.sign(x_dot) * x, 0.0, 2.0 *
    x_thresh)
    avail_accel = force_mag / (masscart + 1e-6) + 1e-3
    v = np.abs(x_dot)
    disc = np.maximum(v * v + 2.0 * avail_accel * remaining, 0.0)
    t_impact = (np.sqrt(disc) - v) / avail_accel
    danger = 1.0 / (t_impact / tau + 1.0)
    out = -A * danger
    return out
```

D.3 The Results on JSSP

We further evaluate JOR on JSSP using a fixed tabu-search solver (Li et al., 2026) with 100k and 400k iterations on DMU instances (Demirkol et al., 1998) spanning eight scales from 20×15 to 50×20. The solver guidance objective is reshaped while the original makespan objective remains unchanged. Table 10 compares JOR against vanilla guidance, Claude-Goal, ShinkaEvolve (Lange et al., 2026), and CORAL (Qu et al., 2026) under identical solver settings and budgets. JOR im-

Table 7: Bayesian optimization results across four evaluation protocols. The reported metric is the normalized optimization cost (↓), combining optimization regret and convergence speed. Lower values indicate better black-box optimization performance. We compare against classical acquisition functions and LLM-based baselines. Results are reported as mean ± standard deviation over benchmark instances; the best result in each row is in bold.

Protocol	T	Classical acquisition functions			LLM-based baselines					JOR (ours)
		EI	UCB (β=2)	MES	Claude-goal	ShinkaEvolve	Eureka	α_{funbo}	CORAL	
OOD-Bench	20	0.872±0.055	0.861±0.063	0.847±0.056	0.872±0.058	0.889±0.050	0.892±0.043	0.860±0.054	0.887±0.045	**0.798±0.056**
	30	0.778±0.048	0.778±0.064	0.757±0.056	0.778±0.054	0.794±0.043	0.796±0.042	0.770±0.050	0.794±0.051	**0.662±0.057**
	50	0.653±0.046	0.668±0.068	0.638±0.049	0.653±0.048	0.662±0.043	0.666±0.049	0.664±0.049	0.662±0.049	**0.511±0.061**
ID-Bench	20	0.748±0.099	0.729±0.114	0.734±0.100	0.702±0.117	0.731±0.116	0.722±0.103	0.730±0.116	0.714±0.111	**0.657±0.122**
	30	0.607±0.116	0.588±0.130	0.606±0.110	0.530±0.121	0.582±0.128	0.560±0.118	0.590±0.125	0.544±0.123	**0.490±0.126**
	50	0.435±0.115	0.426±0.132	0.448±0.111	0.372±0.123	0.393±0.123	0.381±0.116	0.423±0.117	0.356±0.110	**0.308±0.094**
Few-shot Ackley-2D	30	1.179±0.156	1.171±0.159	1.174±0.160	0.790±0.299	1.107±0.126	1.084±0.166	1.180±0.156	0.968±0.326	**0.437±0.152**
HPO-ID AdaBoost	20	0.625±0.130	0.624±0.123	0.621±0.131	0.626±0.133	0.679±0.135	0.659±0.143	0.635±0.133	0.629±0.131	**0.516±0.101**

Table 8: Reward shaping results on CartPole under the BiPaRS protocol (7 out-of-distribution physics configurations × 100 seeds). The reported metric is the average steps per episode over the learning curve (ASPE, ↑; maximum 200) at four training budgets. We compare against standard PPO, representative reward shaping methods, and LLM-based baselines. Results are reported as mean ± standard deviation over the 700 runs; the best result in each row is in bold.

Budget	PPO	Reward shaping			LLM-based baselines					JOR (ours)
		NS	EM	DPBA	Claude-goal	ShinkaEvolve	Eureka	R*	CORAL	
100k	115.2±36.4	155.0±20.6	130.7±29.6	114.9±36.9	168.0±21.5	160.3±20.2	165.6±20.3	159.8±27.6	164.0±21.4	**172.7±22.6**
200k	149.6±28.8	177.1±10.8	164.5±16.2	146.7±34.4	183.2±12.8	179.9±10.4	182.6±10.4	179.1±14.8	181.5±11.3	**186.0±12.8**
400k	172.3±16.8	188.6±5.4	182.2±8.2	164.6±32.6	191.4±7.6	190.0±5.2	191.3±5.2	189.4±7.6	190.7±5.7	**192.9±6.6**
1.2M	190.4±6.2	196.1±2.0	193.7±3.4	167.6±34.5	197.1±2.6	196.5±2.0	197.1±1.8	196.3±3.0	196.8±2.0	**197.6±2.2**

proves most problem sizes, with more consistent gains at 400k iterations; JOR and JOR-CORAL also provide more stable improvements than CORAL across scales. These results show that JOR generalizes beyond routing and effectively improves heuristic guidance for scheduling.

D.4 Reference-Region Alignment Metrics

This section provides the construction of the reference solution region and the alignment metrics used in Sec. 3.3. The goal is to determine whether a reshaped guidance objective induces finite-budget behavior that remains consistent with the high-quality solution region under the original task objective C_0.

During offline objective discovery, the acceptance gate computes the alignment criterion only on a fixed set of alignment instances $\mathcal{I}_{\text{align}}$, which excludes all final test instances. Strong-reference outputs are visible only to the alignment criterion and are not provided to either the structural proposer or zeroth-order numerical optimization. The same metrics may additionally be recomputed on held-out evaluation instances for post-hoc analysis.

For a guidance objective G and task instance x, let

$$\mathcal{S}_G(x) = \{A_{\mathcal{B}}(G; x, \omega_i)\}_{i=1}^{P}, \tag{61}$$

where P denotes the number of repeated finite-budget runs. Let $\mathcal{S}_{C_0}(x)$ denote a high-quality reference solution set constructed using a substantially stronger task-specific procedure operating solely under the original task objective C_0. Alignment is measured by comparing the geometry of the induced solution region $\mathcal{S}_G(x)$ with that of the reference region $\mathcal{S}_{C_0}(x)$.

Construction of the Reference Region. For each task instance x, the reference procedure generates a collection of solutions using only the original task objective C_0. Duplicate outputs are removed and all distinct solutions produced by the reference procedure are retained:

$$\mathcal{S}_{C_0}(x) = \text{Unique}\left(\left\{s_r^{C_0}(x)\right\}_{r=1}^{n_x^{\text{raw}}}\right), \tag{62}$$

Table 9: CVRP results across different solvers, instance sizes, and search budgets. Gap (%, ↓) is always computed under the original routing objective. For VRP200/500/1000, gaps are measured relative to fixed reference costs obtained from vanilla LKH-3 with a budget of 10,000 trials; for CVRPLIB-X (100 instances), gaps are measured relative to the best-known solutions. Gap is reported as mean ± standard deviation over three independent runs, and Time denotes the average single-core solving time per instance in seconds.

Solver	Iteration	Method	VRP200		VRP500		VRP1000		CVRPLib	
			Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
NeuroLKH	100	Vanilla	1.95 ± 0.01	0.74	2.61 ± 0.02	2.09	3.24 ± 0.02	9.76	2.36 ± 0.05	5.47
		Claude-goal	2.01 ± 0.05	0.68	2.66 ± 0.03	2.07	3.27 ± 0.04	9.29	2.27 ± 0.04	6.68
		ShinkaEvolve	1.96 ± 0.06	0.71	2.44 ± 0.01	2.02	2.78 ± 0.05	9.59	2.24 ± 0.10	5.24
		CORAL	2.02 ± 0.08	0.71	2.63 ± 0.02	2.08	3.27 ± 0.05	10.7	2.29 ± 0.08	**5.06**
		JOR-CORAL	1.91 ± 0.03	0.68	2.32 ± 0.03	1.94	2.57 ± 0.02	9.37	$\mathbf{2.05 \pm 0.06}$	5.25
		JOR	$\mathbf{1.78 \pm 0.04}$	**0.56**	$\mathbf{2.28 \pm 0.04}$	**1.91**	$\mathbf{2.54 \pm 0.04}$	**9.26**	2.07 ± 0.06	5.18
	1000	Vanilla	0.62 ± 0.01	6.56	0.81 ± 0.01	16.3	0.85 ± 0.02	47.0	0.70 ± 0.03	34.4
		Claude-goal	0.64 ± 0.02	6.28	0.82 ± 0.04	17.6	0.85 ± 0.02	43.5	0.68 ± 0.03	39.9
		ShinkaEvolve	0.62 ± 0.05	6.00	0.70 ± 0.04	14.6	0.60 ± 0.01	35.9	0.65 ± 0.04	32.4
		CORAL	0.60 ± 0.03	7.05	0.82 ± 0.03	15.8	0.85 ± 0.02	37.8	0.62 ± 0.05	34.4
		JOR-CORAL	0.56 ± 0.05	5.80	0.65 ± 0.03	13.8	$\mathbf{0.55 \pm 0.01}$	32.6	0.61 ± 0.02	31.7
		JOR	$\mathbf{0.50 \pm 0.05}$	**5.17**	$\mathbf{0.60 \pm 0.04}$	**12.6**	$\mathbf{0.55 \pm 0.02}$	**29.2**	$\mathbf{0.60 \pm 0.04}$	**30.0**
	10000	Vanilla	0.04 ± 0.04	77.1	-0.02 ± 0.01	211	-0.25 ± 0.02	590	0.06 ± 0.02	292
		Claude-goal	0.04 ± 0.01	71.3	-0.03 ± 0.01	229	-0.26 ± 0.02	566	0.01 ± 0.04	278
		ShinkaEvolve	0.05 ± 0.01	72.2	-0.10 ± 0.03	191	-0.41 ± 0.01	480	0.05 ± 0.04	268
		CORAL	0.04 ± 0.06	75.5	-0.02 ± 0.02	216	-0.24 ± 0.01	470	0.05 ± 0.04	356
		JOR-CORAL	0.01 ± 0.01	69.5	$\mathbf{-0.13 \pm 0.01}$	182	-0.43 ± 0.02	363	$\mathbf{0.01 \pm 0.00}$	254
		JOR	$\mathbf{-0.03 \pm 0.06}$	**64.6**	$\mathbf{-0.13 \pm 0.03}$	**170**	$\mathbf{-0.44 \pm 0.01}$	**317**	0.02 ± 0.02	**247**
PyVRP	100	Vanilla	4.19 ± 0.06	0.10	4.12 ± 0.04	0.14	3.38 ± 0.04	0.36	4.77 ± 0.05	0.16
		Claude-goal	4.16 ± 0.04	**0.06**	3.98 ± 0.03	**0.10**	3.25 ± 0.05	0.29	4.57 ± 0.04	0.14
		ShinkaEvolve	4.14 ± 0.06	**0.06**	3.97 ± 0.09	**0.10**	3.26 ± 0.02	**0.28**	4.54 ± 0.05	**0.13**
		CORAL	4.23 ± 0.10	**0.06**	4.17 ± 0.04	0.15	3.46 ± 0.04	0.35	4.73 ± 0.09	0.16
		JOR-CORAL	$\mathbf{4.00 \pm 0.06}$	**0.06**	3.89 ± 0.07	0.11	3.25 ± 0.03	0.31	4.55 ± 0.01	**0.13**
		JOR	4.03 ± 0.16	**0.06**	$\mathbf{3.83 \pm 0.02}$	**0.10**	$\mathbf{3.20 \pm 0.03}$	0.30	$\mathbf{4.49 \pm 0.15}$	**0.13**
	1000	Vanilla	2.59 ± 0.05	0.41	2.85 ± 0.02	0.65	2.26 ± 0.01	1.10	3.31 ± 0.02	0.62
		Claude-goal	2.52 ± 0.04	**0.37**	2.79 ± 0.03	0.66	2.21 ± 0.01	0.95	3.24 ± 0.06	0.60
		ShinkaEvolve	2.54 ± 0.03	0.43	2.78 ± 0.01	0.69	2.18 ± 0.01	**0.92**	3.16 ± 0.02	0.62
		CORAL	2.62 ± 0.03	0.42	2.89 ± 0.02	0.67	2.29 ± 0.02	1.10	3.28 ± 0.07	0.67
		JOR-CORAL	$\mathbf{2.43 \pm 0.02}$	0.40	$\mathbf{2.67 \pm 0.04}$	**0.63**	2.13 ± 0.02	0.93	3.18 ± 0.05	**0.59**
		JOR	2.43 ± 0.05	0.40	2.70 ± 0.04	0.63	$\mathbf{2.09 \pm 0.00}$	0.94	$\mathbf{3.13 \pm 0.01}$	**0.59**
	10000	Vanilla	0.07 ± 0.02	3.56	0.85 ± 0.01	5.12	0.90 ± 0.01	7.52	1.48 ± 0.01	4.59
		Claude-goal	0.03 ± 0.03	3.39	0.80 ± 0.01	**4.79**	0.85 ± 0.01	7.19	1.46 ± 0.00	4.49
		ShinkaEvolve	$\mathbf{0.02 \pm 0.05}$	3.48	0.78 ± 0.01	4.89	0.86 ± 0.01	7.17	1.45 ± 0.03	4.61
		CORAL	0.09 ± 0.04	3.57	0.87 ± 0.01	5.25	0.93 ± 0.01	7.53	1.51 ± 0.01	4.78
		JOR-CORAL	$\mathbf{-0.01 \pm 0.03}$	**3.32**	0.74 ± 0.02	4.86	0.79 ± 0.01	6.90	1.43 ± 0.03	**4.36**
		JOR	0.00 ± 0.01	3.44	$\mathbf{0.73 \pm 0.01}$	4.82	$\mathbf{0.80 \pm 0.01}$	**6.79**	$\mathbf{1.42 \pm 0.01}$	4.41

where n_x^{raw} denotes the number of recorded reference outputs before deduplication, and

$$n_x^{\mathrm{ref}} = |\mathcal{S}_{C_0}(x)|$$

denotes the number of distinct retained reference solutions. Consequently, the size of the reference region may vary across task instances.

For combinatorial optimization, the reference region is generated using substantially larger optimization budgets than those assigned to the finite-budget solvers. For CVRP, we run HGS for one hour per instance under the original travel-distance objective using multiple random seeds and retain the distinct final routing solutions. For JSSP, we analogously run OR-Tools for one hour per instance under the original makespan objective using multiple seeds and retain the distinct final schedules.

For EAGLE, each task instance x corresponds to a fixed decoding position of a prompt. The offline trace provides a shared depth-20, beam-10 candidate-node universe together with target-model verification outcomes. To construct a reference tree, we identify the longest target-consistent path in this universe and assign its nodes dominant scores, with ancestors preceding descendants. We then apply the same prefix-closed top-60 selection routine used by the evaluated methods, filling the remaining positions according to the cumulative draft scores. The resulting tree attains the maximum accepted-prefix length available in the recorded candidate universe.

Table 10: JSSP results on DMU benchmark under different search budgets and problem sizes. Gap (%, ↓) is measured under the original makespan objective and reported as mean ± standard deviation over seeds. The best result for each budget and problem size is highlighted in bold.

		Problem Size (Jobs × Machines)							
Budget	**Method**	**20×15**	**20×20**	**30×15**	**30×20**	**40×15**	**40×20**	**50×15**	**50×20**
100k	Vanilla	4.736 ± 0.333	4.875 ± 0.348	5.820 ± 0.559	8.135 ± 0.401	5.378 ± 0.404	7.623 ± 0.482	5.934 ± 0.169	8.046 ± 0.308
	Claude-goal	4.566 ± 0.376	4.781 ± 0.572	5.962 ± 0.225	7.681 ± 0.456	5.417 ± 0.339	7.397 ± 0.387	5.821 ± 0.246	7.819 ± 0.231
	Shinka	4.444 ± 0.275	$\mathbf{4.626 \pm 0.408}$	6.167 ± 0.532	7.937 ± 0.507	5.335 ± 0.335	7.449 ± 0.240	5.773 ± 0.317	7.816 ± 0.620
	CORAL	4.704 ± 0.530	4.702 ± 0.187	5.956 ± 0.404	7.767 ± 0.390	5.232 ± 0.326	7.432 ± 0.405	5.717 ± 0.272	$\mathbf{7.494 \pm 0.388}$
	JOR-CORAL	$\mathbf{4.290 \pm 0.462}$	4.865 ± 0.399	5.887 ± 0.380	7.689 ± 0.472	5.148 ± 0.374	7.338 ± 0.365	5.697 ± 0.224	7.878 ± 0.596
	JOR	4.688 ± 0.396	4.846 ± 0.443	$\mathbf{5.733 \pm 0.362}$	$\mathbf{7.448 \pm 0.538}$	$\mathbf{5.060 \pm 0.295}$	$\mathbf{7.248 \pm 0.334}$	$\mathbf{5.671 \pm 0.164}$	7.753 ± 0.265
400k	Vanilla	2.480 ± 0.104	2.859 ± 0.165	3.699 ± 0.364	4.907 ± 0.258	3.550 ± 0.264	4.180 ± 0.204	4.188 ± 0.007	4.768 ± 0.166
	Claude-goal	2.446 ± 0.079	2.998 ± 0.296	3.500 ± 0.217	4.824 ± 0.215	3.563 ± 0.316	4.248 ± 0.076	3.928 ± 0.346	$\mathbf{4.528 \pm 0.021}$
	Shinka	2.297 ± 0.294	2.758 ± 0.161	3.850 ± 0.192	4.815 ± 0.116	3.566 ± 0.190	4.166 ± 0.359	4.000 ± 0.356	4.688 ± 0.067
	CORAL	2.592 ± 0.411	2.893 ± 0.383	3.629 ± 0.190	4.791 ± 0.331	3.503 ± 0.288	4.227 ± 0.280	4.013 ± 0.052	4.730 ± 0.160
	JOR-CORAL	$\mathbf{2.279 \pm 0.259}$	$\mathbf{2.702 \pm 0.157}$	3.612 ± 0.047	4.683 ± 0.387	3.502 ± 0.248	4.327 ± 0.230	3.935 ± 0.315	4.656 ± 0.197
	JOR	2.296 ± 0.221	2.782 ± 0.159	$\mathbf{3.360 \pm 0.231}$	$\mathbf{4.742 \pm 0.047}$	$\mathbf{3.465 \pm 0.259}$	$\mathbf{4.182 \pm 0.249}$	$\mathbf{3.923 \pm 0.223}$	4.642 ± 0.116

Table 11: Ablation of objective alignment. Fraction of accepted candidates that become harmful under a longer evaluation budget (↓).

Problem	**w/o Align.**	**w/ Align.**
PyVRP	15.38%	**0%**
LLM Inference Acceleration	16.67%	**0%**
BO	28.85%	**3.85%**
RL	22.22%	**0%**

Seed-dependent perturbations of the stepwise draft log probabilities, with standard deviation 0.3, are used to generate reference-region samples while preserving node tokens and parent relationships. We use 10 seeds for the discovery-time alignment gate and 20 seeds for the held-out region study. The distinct reference trees constitute $\mathcal{S}_{C_0}(x)$. Nodes are identified by their indices in the shared candidate universe, and distances are computed only between trees associated with the same decoding position. For discovery-time alignment instances, reference trees are supplied only to the alignment evaluator; held-out evaluation prompts remain completely separate.

For Bayesian optimization, the instance x specifies a black-box function f_x, and the offline evaluator has access to its objective values on a fixed Sobol candidate grid. The reference region $\mathcal{S}_{C_0}(x)$ contains all grid points whose normalized regret is at most 0.01, where regret is normalized using the minimum and maximum objective values on the corresponding grid. If fewer than two points satisfy this threshold, we instead retain the ten grid points with the lowest objective values. The reference region is constructed once and cached independently of the candidate guidance objective. Each evaluated BO run contributes its final best-found query point for comparison with this reference region. Unobserved grid values and reference information are not available through the guidance-function interface. Final test functions are not used to construct discovery-time reference regions.

For reinforcement learning, the reference procedure trains policies directly using the original environment reward without auxiliary reward shaping and with a substantially extended optimization budget. The resulting policies are evaluated solely under the original reward and converted into state-visitation representations using the same fixed initial states and deterministic evaluation protocol as the shaped policies.

Domain-Specific Solution Distance. The alignment metrics require a distance function $d(\cdot, \cdot)$ between two outputs. Because output representations differ across domains, we define a domain-specific distance for each task.

Bayesian Optimization. For BO, each finite-budget output is represented by its best-found query point $y \in [0, 1]^{d_{\mathrm{BO}}}$. Given two output points y_1 and y_2, we use the normalized Euclidean distance

$$d(y_1, y_2) = \frac{\|y_1 - y_2\|_2}{\sqrt{d_{\mathrm{BO}}}}, \tag{63}$$

where $\sqrt{d_{\mathrm{BO}}}$ is the diagonal length of the unit hypercube. This normalization makes the distance scale less sensitive to the dimension of the BO search space.

Reinforcement Learning. For CartPole, each policy is represented by its state-visitation cloud under a fixed deterministic evaluation protocol. Let $\zeta = (p, \dot{p}, \alpha, \dot{\alpha})$ denote a CartPole state, where p is the cart position and α is the pole angle. We normalize the state as

$$\widehat{\zeta} = (p/2.4,\ \dot{p}/3,\ \alpha/0.21,\ \dot{\alpha}/3)\,. \tag{64}$$

Given two state-visitation clouds $\mathcal{C}_1$ and $\mathcal{C}_2$, we use the symmetric Chamfer distance

$$\begin{aligned} d(\mathcal{C}_1, \mathcal{C}_2) = \frac{1}{|\mathcal{C}_1|} \sum_{\zeta \in \mathcal{C}_1} \min_{\zeta' \in \mathcal{C}_2} \|\zeta - \zeta'\|_2 \\ + \frac{1}{|\mathcal{C}_2|} \sum_{\zeta' \in \mathcal{C}_2} \min_{\zeta \in \mathcal{C}_1} \|\zeta' - \zeta\|_2. \end{aligned} \tag{65}$$

We additionally report policy inconsistency using action disagreement on states visited by the reference policies and on a fixed probe-state set.

LLM Inference Acceleration. For EAGLE, each decoding behavior is represented by its verification tree. Let $\mathcal{V}_1$ and $\mathcal{V}_2$ denote the node sets of two verification trees. We use the Jaccard distance

$$d(\mathcal{V}_1, \mathcal{V}_2) = 1 - \frac{|\mathcal{V}_1 \cap \mathcal{V}_2|}{|\mathcal{V}_1 \cup \mathcal{V}_2|}. \tag{66}$$

Since each node uniquely determines a root-to-node path, comparison of node sets also captures differences in the corresponding tree structures. For nearest-neighbor statistics, zero distances caused by identical trees are clipped to $1/120$ to avoid degenerate ratios.

Vehicle Routing Problems. For CVRP, each routing solution is represented by the set of undirected edges appearing in its routes, including depot edges. Let Γ_1 and Γ_2 denote the edge sets of two routing solutions. We define

$$d(\Gamma_1, \Gamma_2) = 1 - \frac{|\Gamma_1 \cap \Gamma_2|}{|\Gamma_1 \cup \Gamma_2|}. \tag{67}$$

This representation makes structurally similar routing solutions close under the metric, as local modifications such as 2-opt or relocate typically alter only a small subset of edges.

Job-Shop Scheduling. For JSSP, each schedule is represented by machine-wise job-ordering sequences. For machine ℓ, let $\mathrm{ord}_\ell^{(1)}$ and $\mathrm{ord}_\ell^{(2)}$ denote the job-ordering sequences of two schedules. Their normalized Kendall–τ distance is

$$d_\ell\left(\mathrm{ord}_\ell^{(1)}, \mathrm{ord}_\ell^{(2)}\right) = \frac{2\, n_{\mathrm{dis}}\left(\mathrm{ord}_\ell^{(1)}, \mathrm{ord}_\ell^{(2)}\right)}{n_\ell(n_\ell - 1)}, \tag{68}$$

where $n_{\mathrm{dis}}(\mathrm{ord}_\ell^{(1)}, \mathrm{ord}_\ell^{(2)})$ is the number of discordant job pairs and n_ℓ is the number of jobs processed on machine ℓ.

Let n_M denote the number of machines. The schedule-level distance is

$$d\left(\mathrm{ord}^{(1)}, \mathrm{ord}^{(2)}\right) = \frac{1}{n_M} \sum_{\ell=1}^{n_M} d_\ell\left(\mathrm{ord}_\ell^{(1)}, \mathrm{ord}_\ell^{(2)}\right). \tag{69}$$

Alignment Tolerance. Candidate acceptance permits a relative alignment tolerance $\epsilon_{\mathrm{align}} = 0.02$:

$$R_{\mathrm{cw}}(G_{\mathrm{new}}; x) \le (1 + \epsilon_{\mathrm{align}}) R_{\mathrm{cw}}(G_{\mathrm{current}}; x). \tag{70}$$

The tolerance accommodates small sampling fluctuations in the estimated reference-region alignment while preventing substantial drift of the induced solution region away from the high-quality region under the original task objective.

Nearest-Neighbor Ratio. In addition to the cross-to-within ratio used for candidate acceptance, we report a complementary nearest-neighbor statistic for post-hoc alignment analysis.

For an output $s \in \mathcal{S}_G(x)$, define its nearest-neighbor distance to the reference region as

$$d_{\mathrm{NN}}^{\mathrm{ref}}(s; x) = \min_{s' \in \mathcal{S}_{C_0}(x)} d(s, s'), \tag{71}$$

and its nearest-neighbor distance within the induced solution region as

$$d_{\mathrm{NN}}^{G}(s;x) = \min_{s' \in \mathcal{S}_G(x) \setminus \{s\}} d(s, s'). \tag{72}$$

The nearest-neighbor ratio is then defined as

$$R_{\mathrm{NN}}(G;x) = \frac{1}{|\mathcal{S}_G(x)|} \sum_{s \in \mathcal{S}_G(x)} \frac{d_{\mathrm{NN}}^{\mathrm{ref}}(s;x)}{d_{\mathrm{NN}}^{G}(s;x)}. \tag{73}$$

Smaller values indicate that the finite-budget outputs lie closer to the reference region relative to their local spacing within the induced solution region. Unlike R_{cw}, the nearest-neighbor ratio is used only as a complementary post-hoc diagnostic and does not enter the candidate acceptance criterion.

D.5 Sensitivity to the Alignment Tolerance.

We evaluate the sensitivity to $\epsilon_{\mathrm{align}}$ on four representative domains—PyVRP, LLM inference acceleration, Bayesian optimization, and reinforcement learning. For each tolerance, we first compute Accepted Harm and Rejected Benefit within each domain and then report their macro-average across the four domains.

Table 12: Sensitivity to the alignment tolerance $\epsilon_{\mathrm{align}}$. Accepted Harm denotes the fraction of accepted candidates that become harmful under a longer evaluation budget, while Rejected Benefit denotes the fraction of rejected candidates that are found to be beneficial.

$\epsilon_{\mathrm{align}}$	Accepted Harm ↓	Rejected Benefit ↓	Mean ↓
1%	0.70	1.91	1.31
2%	0.96	0.89	**0.93**
3%	1.82	0.72	1.26
5%	3.78	0.59	2.19

We study the sensitivity of JOR to the alignment tolerance $\epsilon_{\mathrm{align}}$ by varying it from 1% to 5%. As shown in Table 12, increasing $\epsilon_{\mathrm{align}}$ makes the acceptance criterion less conservative: the fraction of rejected candidates that are later found to be beneficial steadily decreases from 1.91% to 0.59%, while the fraction of accepted candidates that become harmful under a longer evaluation budget increases from 0.70% to 3.78%. Among the tested values, $\epsilon_{\mathrm{align}} = 2\%$ achieves the best overall trade-off, yielding the lowest mean error rate of 0.93%. We therefore use $\epsilon_{\mathrm{align}} = 0.02$ throughout the main experiments.

D.6 Applicability across Coding Agents

Table 13: Applicability of JOR on Claude code and Codex.

Variant	PyVRP			LLM Inference Acceleration		BO	RL
	200-Gap(%) ↓	500-Gap(%) ↓	1000-Gap(%) ↓	MAT ↑	Speedup ↑	Cost ↓	ASPE ↑
Claude Code Sonnet 5	4.03±0.16	**3.83±0.02**	3.20±0.03	**7.60**	**3.72±0.13**	**0.798±0.056**	172.7±22.6
Codex GPT-5.6 sol	**3.93±0.09**	3.87±0.01	**3.17±0.04**	7.50	3.68±0.03	0.751±0.076	**173.9±18.4**

We evaluate the applicability of JOR with two different coding agents, Claude Code (Sonnet 5) and Codex (GPT-5.6 Sol), across four representative tasks: PyVRP, LLM inference acceleration, Bayesian optimization (BO), and reinforcement learning (RL). We evaluate them using the routing gaps after 100 optimization iterations, MAT and inference speedup, optimization cost on OOD-Bench, and ASPE after 100K training steps, respectively. As shown in Table 13, JOR achieves consistently comparable performance with both agents across all four tasks. Codex performs slightly better on the 200- and 1000-node routing instances, and RL ASPE, whereas Claude Code obtains marginally better results on the 500-node instances, LLM inference acceleration and BO. Overall, the small performance differences across agents suggest that the effectiveness of JOR is not tied to a specific coding agent or underlying model, demonstrating its broad applicability across optimization tasks and agent backbones.