RAISE: LLM-BASED AUTOMATED HEURISTIC DESIGN WITH ROBUST ADVERSARY INSTANCE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated Heuristic Design (AHD) with Large Language Models (LLMs) has shown remarkable progress in discovering high-quality heuristics. However, existing LLM-based AHD methods optimize heuristics for a fixed training instance set and may fail catastrophically when deployed under real-world distributional shifts. We propose **R**obust **A**dversary **I**nstance **Se**arch (**RAISE**), a framework that integrates adversarial instance search constrained in a task-specific encoding space into the LLM-based evolutionary search loop. It consists of two loops: an LLM-free inner loop decodes hard instances from an ε-neighborhood of nominal encodings using basis-function perturbations and radial projection, while an LLM-driven outer loop evolves heuristics to optimize aggregate performance over the dynamically accumulated hard-instance set. Experiments span four combinatorial problem families in five task settings, covering five distribution families and 95 test datasets. Existing LLM-based AHD methods degrade by up to $27\times$ under distribution shift, while RAISE consistently attains the best average shifted performance among the reported learned baselines.

1 INTRODUCTION

Designing effective heuristic algorithms for optimization problems has traditionally required extensive domain expertise and manual engineering. Automated Heuristic Design (AHD) seeks to reduce this burden by automatically discovering high-quality heuristic programs, enabling rapid adaptation to new problem domains without much human intervention (Pillay & Qu, 2018; Stützle & López-Ibáñez, 2018; Qu et al., 2020; Zhou et al., 2024; Ma et al., 2025).

The recent integration of Large Language Models (LLMs) into evolutionary search has dramatically advanced AHD (Liu et al., 2024; 2023; Van Stein & Bäck, 2024; Ye et al., 2024; Dat et al., 2025; Zheng et al., 2025; Hu & Zhang, 2025; Ma et al., 2026; Shi et al., 2026; Liu et al., 2026; Novikov et al., 2025; Sun et al., 2026; Gungordu et al., 2026). For example, Liu et al. (2024) propose Evolution of Heuristics (EoH), which uses LLMs as evolutionary operators to iteratively generate, recombine, and refine heuristic functions; Ye et al. (2024) extend this with ReEvo, adding reflective short- and long-term memory; and MCTS-AHD (Zheng et al., 2025) and PartEvo (Hu & Zhang, 2025) integrate Monte Carlo tree search and feature-assisted niche construction to improve search efficiency.

Despite their strong nominal performance (performance on a fixed training instance set), existing LLM-based AHD methods share a critical limitation: they optimize heuristics for a fixed instance set encountered during training, producing highly specialized solutions that may fail catastrophically when deployed under different conditions (Wang et al., 2026; Shi et al., 2026). This robustness gap is particularly problematic in real-world deployment, where problem instances are drawn from unknown or shifting distributions (Zhou et al., 2024; Goh et al., 2025; Wang et al., 2026).

Recent work has attempted to address this. Liu et al. (2026) propose EoH-S, which learns a portfolio of diverse heuristics but relies on a predefined set of diverse training instances; Shi et al. (2026) introduce MoH, a meta-optimization method that still requires explicit multi-task training across manually defined distributions. In both cases the dependence on predefined distributions limits practicality in dynamic or unpredictable environments. Karimi et al. (2025) and Duan et al. (2025) study adversarial instance generation, but impose no constraint on the generated distributions and rely on LLMs to produce instances, which lacks controllability and adds cost.

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.

We propose **RAISE** (**R**obust **A**dversary **I**nstance **S**earch for LLM-based Heuristic Design), which brings constrained hard-instance augmentation into the LLM evolutionary loop. A constrained minimax design goal motivates the search, but RAISE is an empirical robustification heuristic, not a minimax or DRO solver. Its LLM-free inner loop searches against the current best heuristic within a decoded family induced by an encoded-space radius around fixed nominal anchors S_0; its outer loop selects by mean performance on the accumulated instances (Section 2.1). This paper makes the following contributions:

- **Problem formulation.** We state an instance-level constrained minimax design goal over a decoded family with an encoded-space radius constraint. We distinguish this motivation from the implemented hard-instance augmentation and mean-fitness objective, and empirically compare aggregation choices without claiming minimax guarantees.
- **Adversarial instance search.** We propose RAISE, a bi-loop evolutionary framework pairing an LLM-driven outer loop for heuristic evolution with an LLM-free inner loop for hard-instance discovery that uses a basis-function parameterization with encoded-space radial projection to explore the uncertainty set at no additional LLM cost.
- **Empirical validation.** Experiments on four problem families in five task settings, including 95 generated datasets across five distribution families for the online tasks, show that RAISE attains the best shifted-distribution performance among the reported learned baselines, while existing LLM-based AHD methods degrade by up to $27\times$. We quantify the resulting nominal–robustness trade-off explicitly rather than asserting nominal competitiveness.

2 METHODOLOGY

2.1 PROBLEM FORMULATION: ROBUST AUTOMATED HEURISTIC DESIGN

Heuristics, instances, and evaluation. Let $\mathcal{H}$ denote the space of heuristics. Each $h \in \mathcal{H}$ is evaluated on an instance s via a black-box oracle $\mathrm{eval}(h, s) \in \mathbb{R}$, defined so that larger values are better. Standard AHD optimizes $h^* = \arg\max_h \mathbb{E}_{s\sim S_0}[\mathrm{eval}(h, s)]$ under a fixed nominal training set S_0. This is brittle under distributional shift: a heuristic tuned to S_0 may degrade sharply when test instances come from a nearby but different distribution (Wang et al., 2026; Shi et al., 2026; Liu et al., 2026).

Instance representation. Each task uses an encoding $E(s) = \mathbf{x}(s) \in [0, 1]^n$ of scaled instance attributes or generator parameters, not histogram probabilities. OBP stores the ordered item sizes $x_i = a_i/C$; JSP/JSS store scaled operation durations; VRP stores scaled customer demands; TSP stores three scaled spatial-generator parameters (Appendix B.2). Coordinates need not sum to one. For equal-length vectors we use mean absolute coordinate difference,

$$d(\mathbf{x}, \mathbf{y}) = \frac{1}{n}\sum_{j=1}^{n} |x_j - y_j| \le 1.$$

The reported OBP search uses $n = 5{,}000$ item coordinates, not histogram bins; no bin edges or probability normalization are involved.

Uncertainty set. Let $S_0 = \{s_1, \ldots, s_m\}$ be the immutable nominal anchor set, distinct from the growing evaluation set $\mathcal{S}$. Admissible encoded candidates satisfy $\min_{s_i\in S_0} d(\mathbf{x}, E(s_i)) \le \varepsilon$. Let $\mathcal{D}(\mathbf{x}; s_i, \omega)$ denote task-specific decoding using anchor s_i and any generator randomness ω. We write $\mathcal{B}_\varepsilon(S_0) = \{\mathcal{D}(\mathbf{x}; s_i, \omega) : s_i \in S_0,\ \mathbf{x} \in [0, 1]^n,\ d(\mathbf{x}, E(s_i)) \le \varepsilon\}$ for the resulting decoded search family. Adding instances to $\mathcal{S}$ never changes S_0 or this family. The constraint is on the input encoding, not a guarantee about a re-encoded or empirical histogram of the realized instance. This is an engineering surrogate, not distribution-level DRO.

Design goal and implemented objective. The robust AHD *design goal* is the constrained minimax problem

$$h^\star = \arg\max_{h\in\mathcal{H}} \min_{s'\in\mathcal{B}_\varepsilon(S_0)} \mathrm{eval}(h,\, s'), \tag{1}$$

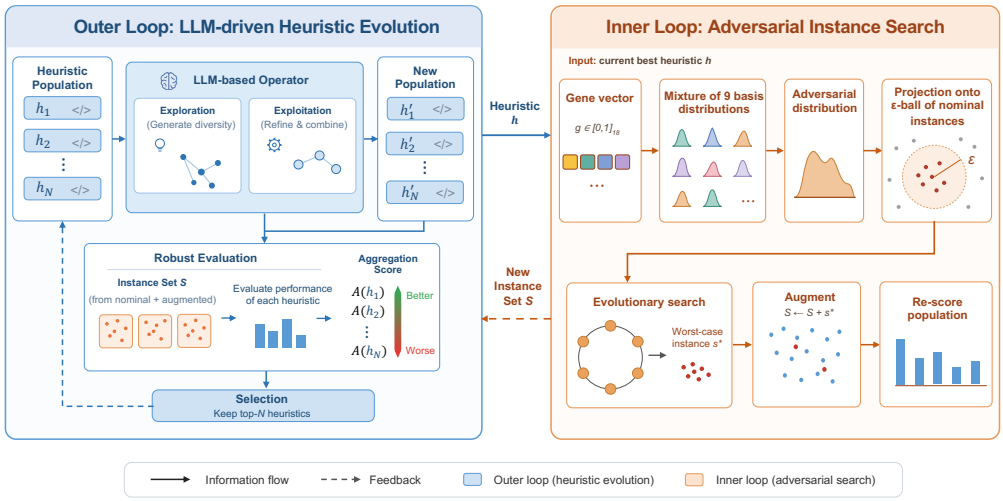


Figure 1: Overview of RAISE: an LLM-driven outer loop for heuristic design coupled with an LLM-free inner search around fixed nominal encodings $E(S_0)$. Only the evaluation set $\mathcal{S}$ grows. The schematic's "distribution" blocks denote feature vectors and basis functions, not probability masses.

which motivates searching for high-loss decoded instances. RAISE does not optimize Equation (1): each refresh searches only against the current best heuristic, not every candidate or a distribution over heuristics. The inner loop adds a decoded hard instance to $\mathcal{S}$, initially S_0, and the outer loop selects heuristics by a weighted aggregate

$$\hat{r}_\lambda(h) \;=\; \sum_{k=1}^{K} \lambda_k \,\mathrm{eval}(h, s'_k), \qquad \lambda \in \Delta^{K-1},\; s'_k \in \mathcal{S}, \tag{2}$$

where strict worst-case fitness on the finite set would instead place all weight on an instance minimizing $\mathrm{eval}(h, s'_k)$ for each h. We use uniform weights $\lambda_k = 1/K$; this does not approximate the minimum over the full decoded family by construction. Section 2.3 motivates this empirical choice, and Appendix E.2 compares worst-case, CVaR, and adaptive weighting schemes.

Role of the radius ε. As $\varepsilon \to 0$, admissible encodings approach $E(S_0)$, but decoded instances need not equal S_0 when decoding is stochastic or lossy. Even with exact decoding, minimax fitness is not nominal mean fitness. Since $d \le 1$, $\varepsilon \ge 1$ removes the radius restriction within the encoded domain, not within all feasible instance space. Intermediate radii control the permitted encoded deviation; Section 3.4 measures the resulting empirical trade-off.

2.2 RAISE Framework Overview

RAISE implements constrained hard-instance augmentation through a bi-loop evolutionary search. Starting from $\mathcal{S} = S_0$ and an initial population, it (i) evolves heuristics with LLM operators in the **outer loop** on the current evaluation set; (ii) every τ generations runs the **inner loop** against the current best heuristic within $\mathcal{B}_\varepsilon(S_0)$; and (iii) augments only $\mathcal{S}$, re-scores the population, and continues (Algorithm 1). The anchors S_0 remain fixed at every refresh. Neither minimax approximation nor convergence guarantees are claimed.

2.3 Outer Loop: LLM-Based Evolutionary Heuristic Search

The outer loop maintains a population $\mathcal{P}$ of heuristic programs and improves it using LLM operators (e.g., synthesize a new heuristic from two high-performing heuristics, or perturb one heuristic to explore its variants); the prompts are given in Appendix B. Each new candidate h is scored by the weighted aggregate (2) over the current instance set $\mathcal{S} = \{s'_1, \ldots, s'_K\}$, comprising nominal and decoded adversarial instances, and candidates are admitted into $\mathcal{P}$ via survival selection retaining the top-$\mathcal{P}_{\max}$ heuristics.

Algorithm 1 RAISE: Robust Adversary Instance Search for LLM-Based Heuristic Design

Require: Task description $\mathcal{T}$, immutable nominal anchors S_0, robustness radius ε, max samples $N_{\max}$, max pop size $\mathcal{P}_{\max}$, refresh interval τ
Ensure: Best heuristic h^*
1: Initialize population $\mathcal{P} \leftarrow \emptyset$, instance set $\mathcal{S} \leftarrow S_0$
2: Sample $\mathcal{P}_{\max}$ heuristics, evaluate on $\mathcal{S}$, initialize $\mathcal{P}$
3: **while** $n_{\text{samples}} < N_{\max}$ **do**
4: **for** each LLM operator **do** ▷ Outer loop
5: Sample parent(s) via rank-weighted selection from $\mathcal{P}$
6: Generate candidate h via the selected LLM operator
7: Evaluate $\hat{r}_\lambda(h)$ by Eq. (2) over the current $\mathcal{S}$
8: Update $\mathcal{P}$ via survival selection
9: **end for**
10: **if** generation $\bmod \tau = 0$ **then** ▷ Inner loop
11: Search against current best $h \in \mathcal{P}$ using fixed anchors S_0; get hard instance s^*
12: Update $\mathcal{S} \leftarrow \mathcal{S} \cup \{s^*\}$
13: Re-score all heuristics in $\mathcal{P}$ on updated $\mathcal{S}$
14: **end if**
15: **end while**
16: **return** $h^* \leftarrow \arg\max_{h \in \mathcal{P}} \hat{r}_\lambda(h)$

Choice of aggregation. Uniform weighting $\lambda_k = 1/K$ is an empirical design choice. Hard-instance augmentation exposes the population to discovered weaknesses, while λ determines the selection signal. Worst-case weighting depends entirely on the single hardest accumulated instance, whose identity can change at refreshes; CVaR averages a small tail. In a separate controlled OBP study at $\varepsilon = 0.1$, averaged over all sizes, capacities, and three seeds, uniform aggregation attains 2.048% shifted waste versus 2.500% for worst-case ($+22\%$) and 2.719% for CVaR ($+33\%$). Adaptive weighting schemes at the same radius improve on uniform weighting by at most 0.05 percentage points and degrade sharply once weight becomes concentrated. These results support the parameter-free mean for the tested average-shift metric, not worst-case optimality over $\mathcal{B}_\varepsilon(S_0)$; the full study is in Appendix E.2.

An instance's influence is moreover not captured by its weight alone: whenever a new adversarial instance enters $\mathcal{S}$ the whole population is re-scored (Algorithm 1, line 13) and survival rankings can change, so even a low-weight instance can eliminate heuristics that fail on it. Instances are never discarded, so $\mathcal{S}$ grows by one per successful refresh, starting from five nominal instances in OBP; S_0 remains unchanged.

2.4 Inner Loop: LLM-Free Adversarial Instance Search

Every τ outer-loop generations, RAISE runs an ***LLM-free*** evolutionary search over encoded candidates against the current best heuristic $h \in \mathcal{P}$. The reported search uses $\mathbf{g} \in [0,1]^{18}$: $g_{1:9}$ are signed perturbation weights and $g_{10:17}$ control basis shapes. The optional length gene g_{18} is inactive in the reported fixed-length OBP sweep.

Basis-function perturbations. A candidate encoding is constructed by adding nine weighted perturbation functions to the mean nominal encoding:

$$\mathbf{x}_{\text{adv}} = \text{clip}\left(\bar{\mathbf{x}} + \sum_{i=1}^{9} w_i \cdot \phi_i(\mathbf{g}),\ 0, 1\right), \tag{3}$$

where $\bar{\mathbf{x}} = |S_0|^{-1} \sum_{s \in S_0} E(s)$ is the fixed mean nominal encoding, $w_i = g_i - 0.5$, and ϕ_i are deterministic vectors evaluated on a coordinate grid: Uniform, Small, Large, Center, Bimodal, Gaussian, Periodic, Poisson-like, and Peak (Appendix B.3). These are perturbation functions, not probability masses or random draws; neither their weights nor the resulting vector are normalized to sum to one. They define a compact search family, not a model of all deployment shifts (Lee et al., 2013). The perturbation at each refresh is discovered by search rather than specified in advance.

Epsilon-boundary projection. For the equal-length encoding, radial projection toward the nearest nominal encoding enforces the encoded-space radius:

$$\mathbf{x}_{\text{proj}} = b + \frac{\varepsilon}{\max(\varepsilon,\, d(\mathbf{x}_{\text{adv}}, b))}(\mathbf{x}_{\text{adv}} - b), \qquad b = \arg\min_{\mathbf{z}\in E(S_0)} d(\mathbf{x}_{\text{adv}}, \mathbf{z}), \tag{4}$$

where $E(S_0) = \{E(s_i) : s_i \in S_0\}$ never includes newly generated instances. For $\varepsilon > 0$, the result remains in $[0,1]^n$ and satisfies $\min_{s_i\in S_0} d(\mathbf{x}_{\text{proj}}, E(s_i)) \le \varepsilon$ at every refresh; candidates already inside the ball are unchanged. An audit of the saved OBP runs confirms unchanged anchors and no encoded-radius violations (Appendix B.2). No probability normalization follows projection. Task-specific decoding can round values or regenerate geometry, so this bound does not generally transfer to $d(E(s'), b)$ after decoding. Among the reported component-removal variants, the variant without projection has the highest shifted waste (Section 4).

Inner evolutionary algorithm. The inner search runs G_{in} generations on P_{in} gene vectors initialized uniformly in $[0,1]^{18}$. It constructs and projects candidates via (3)–(4), checks encoded-space feasibility, and evaluates $s' = \mathcal{D}(\mathbf{x}_{\text{proj}}; s_i, \omega)$ using $-\text{eval}(h, s')$. For OBP, decoding directly sets $a'_j = Cx_{\text{proj},j}$ rather than sampling from a probability distribution. Truncation selection keeps the top half; uniform crossover ($p_{\text{cross}} = 0.5$) and Gaussian mutation ($\sigma = 0.12$, per-gene probability 0.35) produce new genes clipped to $[0,1]$. The worst-scoring decoded instance from an admissible encoding is appended to $\mathcal{S}$. Algorithm 2 details this procedure.

3 Experimental Studies

3.1 Experimental Setup

Tasks. **1) Online Bin Packing (OBP):** n items arrive sequentially into bins of capacity C; the objective is the waste ratio $(\text{bins used} - \text{lower bound})/\text{lower bound}$. Trained on 5 Weibull instances (Romera-Paredes et al., 2024; Liu et al., 2024), tested with $C \in \{100, 200, 300, 400\}$, $n \in \{1000, 5000, 10000\}$, and 5 item-size distributions (Uniform, Normal, Lognormal, Exponential, Triangular) per size, giving $4 \times 3 \times 5 = 60$ datasets. **2) Online Job Shop Scheduling (JSP):** jobs with stochastic processing times are assigned to machines online, minimizing normalized makespan; trained on 5 uniform instances with m=10 machines and j=20 jobs, tested on $m \in \{10, 20\}$, $j \in \{20, 50\}$ with 5 distributions each (20 datasets). **3) Online Vehicle Routing (VRP):** customers with stochastic demands arrive online, minimizing the inverse composite routing score; trained on 5 instances with v=10 vehicles and 50 customers, tested on $v \in \{5, 10, 15\}$ with 5 distributions each (15 datasets). **4–5) Offline tasks:** to test whether the mechanism is tied to online settings we additionally evaluate offline job-shop scheduling (JSS) and offline TSP with Guided Local Search (Voudouris et al., 2018) on 47 real TSPLIB instances (Reinelt, 1991). All task and distribution details are in Appendix C.

Baselines. *Classical:* BestFit, FirstFit (OBP); SPT, MinSlack, EDD (JSP); Nearest Feasible, Slack-Preserving, and Urgency-Weighted Insertion (VRP). *LLM-based AHD:* **EoH** (Liu et al., 2024), **ReEvo** (Ye et al., 2024), **PartEvo** (Hu & Zhang, 2025), each optimized on the nominal training set. *Robustness-aware:* **EoH-S** (Liu et al., 2026) (reported heuristic trained on 128 diverse instances) and **MoH** (Shi et al., 2026) (heuristic as reported in the original paper). *Specialized learned policy:* an online **DRL** policy adapted from the official implementation of Zhao et al. (2021) to our 1D setting, trained and evaluated on the same distributions as all other methods.

Settings. All LLM-driven methods use 1,000 samples, population size 10, and GPT-5-mini as backbone, and train on the 5 nominal instances except EoH-S (128 diverse instances, as in its paper). RAISE uses $\tau = 5$, $P_{\text{in}} = 8$, $G_{\text{in}} = 4$; we ablate $\varepsilon \in \{0.02, 0.05, 0.1, 0.2\}$ on OBP and retain $\varepsilon = 0.1$ for other tasks. Hyperparameters and prompts are in Appendices C and B.

3.2 Results Across Different Distributions

Figure 2 summarizes the results on OBP; Table 1 reports averages over all twelve size configurations and Table 16 (Appendix D.1) the per-distribution breakdown at n=5000, C=200.

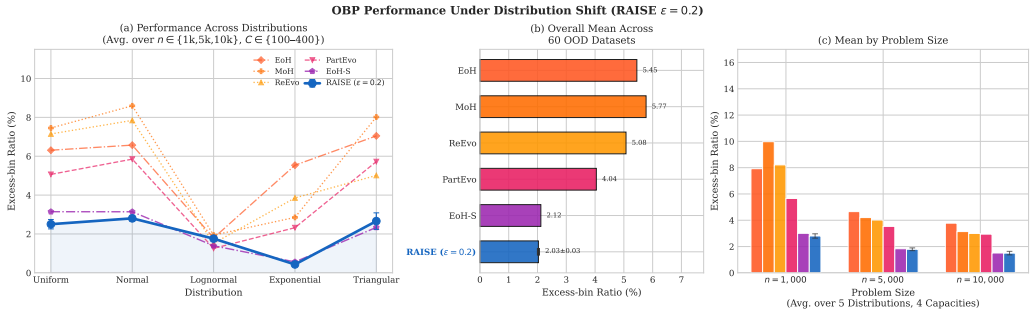


Figure 2: Online bin packing excess-bin ratio (%, lower is better) under distribution shift. RAISE uses $\varepsilon = 0.2$; its values are means over three independent runs and its error bars denote the sample standard deviation across runs. Baselines are fixed reported heuristics. **(a)** Results by distribution, averaged over all sizes and capacities. **(b)** Overall mean across all 60 OOD datasets. **(c)** Results by problem size, averaged over five distributions and four capacities.

Table 1: Average waste ratio (%) over five out-of-distribution test sets (Uniform, Normal, Lognormal, Exponential, Triangular) on Online Bin Packing across twelve size configurations ($n \in \{1\text{k}, 5\text{k}, 10\text{k}\}$, $C \in \{100, 200, 300, 400\}$). Each RAISE entry is the mean of three independent runs. **Bold**: best per column; underline: second best. Lower is better.

Method	$C = 100$			$C = 200$			$C = 300$			$C = 400$		
	1k	5k	10k	1k	5k	10k	1k	5k	10k	1k	5k	10k
BestFit	2.645	1.882	1.608	**2.636**	1.948	1.675	**2.656**	1.984	1.725	**2.644**	2.006	1.732
FirstFit	3.570	2.527	2.147	3.629	2.644	2.287	3.633	2.691	2.341	3.647	2.714	2.367
EoH	6.272	3.799	3.037	8.137	4.776	3.829	8.511	4.957	4.092	8.761	5.093	4.158
ReEvo	5.457	2.591	1.827	7.880	3.790	2.878	9.322	4.630	3.486	10.187	5.053	3.801
PartEvo	4.921	2.453	2.155	5.093	3.321	2.885	6.001	3.922	3.208	6.613	4.434	3.527
EoH-S	2.766	**1.741**	**1.428**	2.969	1.828	1.513	3.091	1.865	1.531	3.203	1.899	1.563
MoH	6.217	2.632	1.937	9.310	3.856	2.867	11.355	4.685	3.550	13.016	5.640	4.199
RAISE ($\epsilon = 0.02$)	5.424	2.611	2.045	6.686	3.086	2.420	7.380	3.446	2.603	8.112	3.773	2.808
RAISE ($\epsilon = 0.05$)	2.773	1.752	1.444	3.089	1.868	1.535	3.362	1.980	1.623	3.623	2.108	1.714
RAISE ($\epsilon = 0.1$)	2.649	1.797	1.523	2.705	1.858	1.594	2.745	1.864	1.623	2.694	1.882	1.639
RAISE ($\epsilon = 0.2$)	**2.598**	1.791	1.504	2.702	**1.769**	**1.489**	2.857	**1.781**	**1.488**	3.060	**1.831**	**1.519**

LLM-based methods collapse under distribution shift. At n=5000, C=200, EoH reaches 5.182% waste on Uniform and 6.308% on Normal, ReEvo 5.466% / 6.539%, both far above classical BestFit (1.717% / 2.803%). MoH, despite meta-level training, similarly collapses (5.277% / 6.531%), as it still relies on multi-task training over predefined distributions. Heuristics optimized on a fixed nominal distribution are thus highly specialized and brittle.

RAISE achieves consistent OOD robustness. Across all 60 OOD datasets, RAISE$^{\varepsilon=0.2}$ attains an average shifted waste of 2.032%, compared with 2.116% for the strongest robustness-aware baseline, EoH-S. At n=5000, C=200, the corresponding averages are 1.769% and 1.828%; on Exponential, RAISE reaches 0.382% against EoH's 5.303%, a 13.9× reduction. EoH-S additionally needs 128 diverse training instances, whereas RAISE uses only 5 nominal instances, a distinction that matters when shifted training data are unavailable.

Robustness generalizes across problem scales. RAISE's advantage is consistent across the twelve size configurations (Table 1): the primary $\varepsilon = 0.2$ variant beats every non-RAISE learned baseline on 10 of 12 configurations and ranks second among the non-RAISE learned baselines on the remaining two. Classical BestFit remains strong at $n = 1$k, where small-instance packing is easier, but RAISE surpasses it throughout $n \geq 5$k. The gains scale with instance size, as expected if adversarial search becomes more informative as scale grows.

Table 2: Performance on online JSP (normalized makespan) and online VRP (inverse composite score). Lower is better. JSP columns average over 10 configurations (5 processing-time distributions $\times$ 2 job counts: 20 or 50 jobs); VRP columns average over 5 demand distributions. Classical baselines are task-specific (— = not applicable). **Bold**: best per column; underline: second best.

	Online JSP (↓)		**Online VRP** (↓)		
Method	**10 machines**	**20 machines**	**5 vehicles**	**10 vehicles**	**15 vehicles**
SPT	1.3561	1.3956	—	—	—
MinSlack	1.3812	1.4346	—	—	—
EDD	1.4233	1.4886	—	—	—
NF-Insert	—	—	0.9347	0.9748	1.0028
SP-Insert	—	—	0.8971	0.9360	0.9665
UW-Insert	—	—	0.9000	0.9346	0.9670
EoH	1.2284	1.2539	0.9017	0.9273	**0.9561**
ReEvo	1.2421	1.2500	0.8966	0.9287	0.9613
RAISE	**1.2266**	**1.2390**	**0.8931**	**0.9263**	0.9570

3.3 Results on Online JSP and VRP

Table 2 reports OOD performance on JSP and VRP. RAISE achieves the lowest normalized makespan on both machine scales (1.2266 / 1.2390) against EoH (1.2284 / 1.2539) and ReEvo (1.2421 / 1.2500), with the advantage widening at 20 machines where the richer combinatorial structure amplifies the benefit of adversarial exposure during search. On VRP it attains the best inverse composite score at 5 vehicles (0.8931) and 10 vehicles (0.9263) and is marginally behind EoH at 15 vehicles (0.9570 vs. 0.9561). Differences among LLM-based methods are smaller on OVRP than on OBP, suggesting that OVRP's combinatorial structure already provides some inherent distributional diversity across the tested families; RAISE ranks first in two fleet-size configurations and second in the third.

3.4 Nominal–Robustness Trade-off

Robustness is not free, and we measure what it costs. Table 3 reports performance on the five nominal Weibull training instances alongside the average over the five shifted families, together with a task-specific online DRL policy (Zhao et al., 2021). First, the nominal-optimized LLM-based methods are excellent on the nominal set (0.139–0.278%) and lose an order of magnitude under shift (14–27×). Second, the primary RAISE setting transfers much more evenly ($1.485\% \rightarrow 1.769\%$, or $1.2\times$); its nominal performance is close to classical BestFit (1.551%) but clearly below the nominal-optimized methods. This is the trade-off ε controls: with $\varepsilon = 0.02$, the new three-run evaluation improves nominal performance to 0.484% but increases shifted waste to 3.086%, whereas $\varepsilon = 0.2$ gives the best shifted average. As $\varepsilon \rightarrow 0$ the objective reduces to standard AHD, where the deployment distribution is known a nominal-optimized method remains preferable, and RAISE targets the case where it is not. Third, the specialized online DRL policy is a substantially stronger robustness baseline than the general-purpose LLM-AHD methods, where every reported checkpoint beats EoH, ReEvo, PartEvo, and MoH under shift, although EoH-S remains better than the DRL checkpoints, yet RAISE still attains the lowest average shifted ratio (1.769% vs. 2.105%) (Appendix E.1).

3.5 Beyond Online Optimization

The inner loop requires only a black-box eval, so it is not tied to online decision-making. We attach RAISE unchanged to two offline pipelines: offline job-shop scheduling, where it evolves the scheduling heuristic, and offline TSP, where it evolves only the guide heuristic inside a classical, non-learned Guided Local Search solver (Voudouris et al., 2018) on 47 real TSPLIB instances (Reinelt, 1991). Table 4 compares both against the identical pipeline with the inner loop removed (i.e., standard LLM-driven AHD), which isolates the contribution of constrained adversarial search. On offline JSS the inner loop improves the makespan ratio on all five families; on TSPLIB the mean optimality gap improves $0.5447\% \rightarrow 0.4932\%$, best in every size band, against 3.3565% for classical GLS. The TSPLIB geometry was never sampled from any training-time generator, so the shift is real rather than synthetic (Appendix E.3).

Table 3: Nominal versus shifted performance on online bin packing (n=5000, C=200), excess-bin ratio (%, lower is better). *Nominal*: the five nominal Weibull training instances; *shifted*: average over the five OOD families; *degradation*: shifted/nominal. The RAISE row averages three independent runs. Full per-checkpoint results in Appendix E.1.

Method	**Nominal**	**Avg. shifted**	**Degradation**
EoH	0.199	4.776	24.0×
ReEvo	**0.139**	3.790	27.3×
PartEvo	0.199	3.321	16.7×
MoH	0.278	3.856	13.9×
DRL, specialized online policy (best ckpt.)	0.634	2.105	3.3×
RAISE	1.485	**1.769**	**1.2×**

Table 4: Offline tasks, both using $\varepsilon = 0.1$ and compared against the identical pipeline with the inner loop removed. *Top*: offline job-shop scheduling, mean makespan ratio (↓) per distribution family, averaged over all problem configurations. *Bottom*: offline TSP with Guided Local Search on 47 real TSPLIB instances, mean optimality gap (%, ↓) grouped by number of cities. **Bold**: best per column.

(a) Offline job-shop scheduling — makespan ratio ↓

Method	Uniform	Normal	Lognormal	Exponential	Triangular	**Avg.**
RAISE w/o inner loop	1.2507	1.2660	1.2620	1.2480	1.2412	1.2536
RAISE	**1.1939**	**1.2315**	**1.2254**	**1.2031**	**1.2107**	**1.2129**

(b) Offline TSP on 47 real TSPLIB instances — optimality gap (%) ↓

Method	50–100 (n=6)	100–200 (n=21)	200–1000 (n=20)	**Avg.**
GLS (classical)	2.9920	3.4281	3.3906	3.3565
RAISE w/o inner loop	0.5336	0.1787	0.9323	0.5447
RAISE	**0.4848**	**0.1602**	**0.8454**	**0.4932**

4 Ablation Study

We report three component-removal variants: (i) **w/o Robust Search** drops the adversarial inner loop, reducing RAISE to standard AHD; (ii) **w/o Basis Functions** replaces the nine-basis parameterization with flat-vector crossover and Gaussian mutation; and (iii) **w/o ε-Mapping** keeps the parameterization but skips the boundary projection.

Table 5 provides diagnostic evidence that each retained ablated variant is less robust on average than the current full model. The variant without ε-mapping is worst (5.96% average and 10.23% on Triangular), followed by the flat-vector variant without the basis functions (3.98%) and the variant without robust search (3.60%).

Aggregation, weighting, and nominal distribution. Three further studies support Section 2.3 and probe dependence on the nominal set. *Aggregation:* uniform mean attains 2.048% shifted waste versus 2.500% (worst case) and 2.719% (CVaR), best on four of five families. *Adaptive weighting:* the best of seven Hedge/age-decay configurations reaches 2.000%, only 0.05 points better than uniform, while an aggressive age decay ($\gamma = 0.5$) collapses to 4.909% shifted and 23.870% nominal—mild adaptivity helps marginally, concentration hurts badly. *Nominal sensitivity:* the Weibull, Normal, and Exponential nominal runs attain 2.048%, 1.985%, and 2.848% over all 60 OOD datasets; all outperform the standard fixed-nominal AHD baselines, while only the Weibull and Normal runs outperform EoH-S (2.116%). Full tables and definitions are in Appendix E.

5 Discussion

What the minimax formulation is and is not. Equation (1) is a motivating design goal, not RAISE's optimized objective. The inner loop attacks only the current best heuristic; it neither computes an adversarial response for every candidate nor optimizes against a distribution over

Table 5: Ablation study on Online Bin Packing. Shifted waste ratio (%) under five out-of-distribution test settings, averaged over all problem sizes ($n \in \{1000, 5000, 10000\}$) and bin capacities ($C \in \{100, 200, 300, 400\}$). Lower is better; **bold** marks the best result per column.

Method	Waste Ratio (%) by Distribution					Avg.
	Uniform	Normal	Lognormal	Exponential	Triangular	
w/o Robust Search	4.94	5.32	**1.40**	1.21	5.11	3.60
w/o Basis Functions	4.49	4.84	2.77	2.53	5.28	3.98
w/o ε-Mapping	7.16	8.40	1.67	2.33	10.23	5.96
RAISE (full)	**2.50**	**2.80**	1.76	**0.43**	**2.66**	**2.03**

heuristics. The outer loop uses mean fitness on the accumulated set, not a worst-case objective. RAISE is therefore constrained hard-instance augmentation with empirical robustness benefits, without a minimax approximation or DRO guarantee. The aggregation study (Appendix E.2) supports mean fitness on one OBP setting's average-shift metric; it does not establish minimax robustness.

Robustness has a price, and ε is the price tag. Section 3.4 makes the trade-off explicit rather than claiming uniform dominance. For example, on online bin packing, RAISE gives up roughly an order of magnitude of nominal performance and buys near-invariance under the tested shifts. We therefore scope the central claim as: constrained hard-instance augmentation improves performance under the tested distribution shifts, not that RAISE dominates every specialized method on every criterion.

The search-space prior is a real assumption. RAISE removes the need to enumerate deployment distributions in advance, as EoH-S and MoH require, but not the need for a domain-appropriate instance representation and perturbation family; where a meaningful feature axis or basis family is hard to specify, the present instantiation would need extra design work. The ablation reports a cross-setting gap (2.03% for the full model versus 3.98% without the basis parameterization), so the nine-basis mixture is an interpretable instantiation, not a domain-independent solution.

Relation to co-evolution and instance generation. RAISE is an alternating algorithm–instance process, related to co-evolutionary search, adversarial instance generation, hard-example mining, and curriculum-style generation (Appendix A). Two properties distinguish it: candidate encodings are constrained to an ε-neighborhood of the immutable nominal encodings $E(S_0)$, independently of the growing evaluation set; and the inner loop is LLM-free, so robustification costs no extra API calls ($\approx$ \$1.79 vs. $\approx$ \$1.78 for EoH at an identical 1,000-call budget; Appendix G).

6 Conclusion

We presented RAISE, a bi-loop evolutionary framework for constrained hard-instance augmentation in LLM-based AHD. An LLM-driven outer loop evolves heuristics by mean fitness while an LLM-free inner loop searches against the current best heuristic using a fixed encoded-space neighborhood of the nominal anchors. This provides empirical robustification without claiming to solve the motivating minimax problem. Across four problem families in five task settings, including 95 generated online datasets, existing LLM-based AHD methods degrade by up to $27\times$ under shift while RAISE attains the best average shifted performance among the reported learned baselines.

Limitations and future work. RAISE models single-dimensional shifts, depends on a hand-designed instance representation and perturbation family, and trades nominal performance for robustness; replacing the nine-basis mixture with a learned perturbation model is the most direct extension.

AI USE STATEMENT

LLMs are a core component of the proposed heuristic-search method, as described in Section 2. We also used generative AI tools to assist with writing. The authors take responsibility for verifying all AI-assisted content and for the final scientific claims.

REFERENCES

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26931–26938, 2025.

Ruibo Duan, Yuxin Liu, Xinyao Dong, and Chenglin Fan. Ealg: Evolutionary adversarial generation of language model-guided generators for combinatorial optimization. *arXiv preprint arXiv:2506.02594*, 2025.

Yong Liang Goh, Zhiguang Cao, Yining Ma, Jianan Zhou, Mohammed Haroon Dupty, and Wee Sun Lee. SHIELD: Multi-task multi-distribution vehicle routing solver with sparsity and hierarchy. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=6DJEaz1cCj.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Oguzhan Gungordu, Siheng Xiong, and Faramarz Fekri. Pathwise: Planning through world model for automated heuristic design via self-evolving LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=aiOv6NhF3N.

Qinglong Hu and Qingfu Zhang. Partition to evolve: Niching-enhanced evolution with llms for automated algorithm discovery. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of artificial intelligence research*, 36:267–306, 2009.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

J. Q. James, Wen Yu, and Jiatao Gu. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20 (10):3806–3817, 2019.

Pantea Karimi, Dany Rouhana, Pooria Namyar, Siva Kesava Reddy Kakarla, Venkat Arun, and Behnaz Arzani. Robust heuristic algorithm design with llms. *arXiv preprint arXiv:2510.08755*, 2025.

Carl Lee, Felix Famoye, and Ayman Y. Alzaatreh. Methods for generating families of univariate continuous distributions in the recent decades. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5(3):219–238, 2013.

Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249*, 2023.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, ICML 2024, 2024.

Fei Liu, Yilu Liu, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37090–37098, 2026.

Renke Liu, Rajesh Piplani, and Carlos Toro. A spatiotemporal graph neural network based deep reinforcement learning method for online distributed job-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2025.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

Zeyuan Ma, Hongshu Guo, Yue-Jiao Gong, Jun Zhang, and Kay Chen Tan. Toward automated algorithm design: A survey and practical guide to meta-black-box-optimization. *IEEE Transactions on Evolutionary Computation*, 2025.

Zeyuan Ma, Yue-Jiao Gong, Hongshu Guo, Jiacheng Chen, Yining Ma, Zhiguang Cao, and Jun Zhang. Llamoco: Instruction tuning of large language models for optimization code generation. *IEEE Transactions on Evolutionary Computation*, 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Nelishia Pillay and Rong Qu. *Hyper-heuristics: theory and applications*. Springer, 2018.

Rong Qu, Graham Kendall, and Nelishia Pillay. The general combinatorial optimization problem: Towards automated algorithm design. *IEEE Computational Intelligence Magazine*, 15(2):14–23, 2020.

Hamed Rahimian and Sanjay Mehrotra. Distributionally robust optimization: A review. *arXiv preprint arXiv:1908.05659*, 2019.

Gerhard Reinelt. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4): 376–384, 1991.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Shiori Sagawa*, Pang Wei Koh*, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ryxGuJrFvS`.

Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Generalizable heuristic generation through LLMs with meta-optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=tIQZ7pVN6S`.

Thomas Stützle and Manuel López-Ibáñez. Automated design of metaheuristic algorithms. In *Handbook of metaheuristics*, pp. 541–579. Springer, 2018.

Weiwei Sun, Shengyu Feng, Shanda Li, and Yiming Yang. Co-bench: Benchmarking language model agents in algorithm search for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33126–33134, 2026.

Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29(2): 331–345, 2024.

Christos Voudouris, Edward P. K. Tsang, and Abdullah Alsheddy. Guided local search. In *Handbook of Metaheuristics*, pp. 321–361. Springer, 2018.

Rongzheng Wang, Yihong Huang, Muquan Li, Jiakai Li, Di Liang, Bob Simons, Pei Ke, Shuang Liang, and Ke Qin. Rethinking llm-driven heuristic design: Generating efficient and specialized solvers via dynamics-aware optimization. *arXiv preprint arXiv:2601.20868*, 2026.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. Online 3D bin packing with constrained deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 741–749, 2021.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Do1OdZzYHr.

Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, pp. 42769–42789, 2023.

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

APPENDIX

TABLE OF CONTENTS

A Related Work **14**

A.1 LLM-Based Automated Heuristic Design 14

A.2 Hyper-Heuristics and Automated Algorithm Configuration 14

A.3 Adversarial Training and Minimax Learning 15

B Detailed Method Description **15**

B.1 Bi-loop Search Architecture 15

B.2 Task-Specific Encodings and Decoding 16

B.3 Inner Adversarial Instance Search 16

B.4 LLM Evolutionary Operators 19

C Detailed Experimental Design and Settings **20**

C.1 Benchmark Tasks and Instance Generation 20

C.2 Baseline Algorithm Descriptions 24

C.3 RAISE Hyperparameter Settings 26

D Comprehensive Experimental Results **26**

D.1 Online Bin Packing (OBP) 26

D.2 Online Job Shop Scheduling (JSP) 26

D.3 Online Vehicle Routing (VRP) 28

E Extended Discussion and Additional Studies **28**

E.1 Nominal Performance and Comparison with a Specialized Online Policy 29

E.2 Aggregation and Instance Weighting in the Outer Loop 30

E.3 Offline Tasks: Job-Shop Scheduling and TSP on TSPLIB 32

E.4 Convergence Behaviour of the Bi-Level Search 33

E.5 Sensitivity to the Nominal Training Distribution 34

E.6 Scope of the Search-Space Prior 35

F RAISE Heuristic Functions: Code and Robustness Analysis **36**

F.1 Online Bin Packing 36

F.2 Online Job Shop Scheduling 37

F.3 Online Vehicle Routing 39

F.4 Offline Job Shop Scheduling 40

F.5 Offline Traveling Salesperson Problem with Guided Local Search 43

G Computational Cost Analysis **45**

G.1 RAISE Cost Breakdown 45
G.2 Comparison with Existing Frameworks 45

A Related Work

A.1 LLM-Based Automated Heuristic Design

The explosion of capable LLMs has unlocked a new paradigm for AHD in which the LLM serves as a black-box mutation and crossover operator over algorithm space. EoH (Liu et al., 2024) is one of the preliminary works on LLM-based AHD. It maintains a population of heuristic Python functions, each scored on a fixed set of problem instances, and uses four LLM prompt types to create, crossover, and mutate programs. EoH achieves state-of-the-art performance on online bin packing and traveling salesman problems under nominal distribution conditions. ReEvo (Ye et al., 2024) extends EoH with a dual-memory architecture: a short-term memory of recent high-scoring programs and a long-term memory of reflective insights generated by the LLM analyzing performance gaps. ReEvo shows that guided memory injection substantially improves sample efficiency, achieving competitive results with fewer LLM calls. HSEvo (Dat et al., 2025) proposes a harmony search framework to balance exploitation and exploration. MCTS-AHD (Zheng et al., 2025) employs Monte Carlo tree search to organize heuristics in a tree structure, enabling more effective exploration of the search space. PartEvo (Hu & Zhang, 2025) incorporates feature-assisted niche construction within abstract search spaces, enabling the seamless integration of niche-based search strategies from evolutionary computation.

Generalizable heuristic design EoH-S (Liu et al., 2026) introduces portfolio-based AHD: instead of seeking a single best heuristic, it discovers a diverse set of complementary heuristics with varying strengths. EoH-S is particularly relevant as a baseline in our distribution-shift experiments, as maintaining a diverse portfolio implicitly confers some cross-distribution robustness. MoH (Shi et al., 2026) leverages LLMs to iteratively refine a meta-optimizer that autonomously constructs diverse heuristic-optimizers through self-invocation, thereby eliminating reliance on a predefined evolutionary computation heuristic-optimizer. These constructed heuristic-optimizers subsequently evolve heuristics for downstream tasks, enabling broader heuristic exploration. MoH further employs a multi-task training scheme to promote generalization.

A key limitation shared by all of the above methods is that they optimize for a *fixed* nominal instance distribution. While EoH-S and MoH propose different strategies to enhance generalization, both still rely on predefined instance sets and offer no guarantee of robust performance under unknown distributions. RAISE explicitly addresses this limitation via distributional robustness through adversarial instance search.

Adversarial instance search. Karimi et al. (2025) employ an LLM to identify regions of the input space where a heuristic underperforms and to suggest targeted improvements. Duan et al. (2025) propose a co-evolutionary framework that jointly evolves both the heuristic algorithm and an instance generator using LLMs, with dynamically generated instances serving to enhance heuristic robustness throughout the evolutionary process.

While both approaches incorporate adversarial search into LLM-driven AHD, they impose no explicit constraints on the instance distribution and rely on LLMs to generate adversarial instances directly, which limits controllability and incurs additional inference cost. In contrast, RAISE employs a *constrained* adversarial search that bounds the divergence between adversarial and nominal instance distributions, striking a principled balance between worst-case robustness and performance on the nominal distribution. Crucially, the worst-case instance search loop in RAISE is LLM-free, making it lightweight and ready to work with a broad range of LLM-driven AHD frameworks.

A.2 Hyper-Heuristics and Automated Algorithm Configuration

Burke et al. (2013) survey hyper-heuristics across two dimensions: (1) heuristic selection vs. heuristic generation, and (2) constructive vs. perturbative. RAISE belongs to the *generative, constructive*

quadrant—it synthesizes new heuristic functions from scratch rather than selecting or composing existing low-level heuristics.

Traditional hyper-heuristics rely on hand-crafted low-level heuristic libraries and meta-level selectors (e.g., simulated annealing, reinforcement learning) to adaptively combine them. This approach is interpretable and efficient but fundamentally limited by the quality of the low-level library.

Algorithm configuration methods such as SMAC (Hutter et al., 2011), ParamILS (Hutter et al., 2009), and irace (López-Ibáñez et al., 2016) find optimal hyperparameter settings for fixed algorithm templates using model-based search. They operate in continuous or mixed-integer parameter spaces, enabling principled exploration, but they cannot *design* new algorithmic logic.

A.3 Adversarial Training and Minimax Learning

DRO (Rahimian & Mehrotra, 2019) is a classical framework in operations research and machine learning for decision-making under distributional uncertainty. The key idea is to replace the expected objective $\mathbb{E}_{P_0}[f(x)]$ with the worst-case expected objective $\sup_{P\in\mathcal{U}} \mathbb{E}_P[f(x)]$ over an ambiguity set $\mathcal{U}$.

Adversarial training (Goodfellow et al., 2014) and minimax learning originate in the GAN framework, where a generator and discriminator are co-trained via a minimax objective. The connection to distributional robustness was made explicit in subsequent work: training on adversarially perturbed data provably improves worst-case generalization under the assumed perturbation model.

Domain adaptation methods (Ben-David et al., 2010) study the theoretical conditions under which a model trained on a source distribution generalizes to a target distribution, providing H-divergence-based bounds on transfer error. These bounds motivate training procedures that minimize distribution shift, aligning with RAISE's adversarial instance generation objective.

Distributionally robust neural networks (Sagawa* et al., 2020) apply group DRO to improve worst-case accuracy across subgroups in neural network training. RAISE shares the motivation of improving performance under shift, but uses a different objective: an evolutionary inner loop discovers hard instances against the current best program, and outer selection uses uniform mean fitness on the accumulated set, rather than worst-case group weighting.

RAISE also uses *black-box* heuristic evaluation and derivative-free instance search. These connections are motivational; no minimax or DRO guarantee is transferred to the implemented augmentation procedure.

B Detailed Method Description

This appendix provides comprehensive documentation of RAISE's components beyond what is possible in the main paper. We describe each component in detail, provide complete pseudocode for sub-routines, document all LLM prompt templates, give a full hyperparameter table, and discuss implementation details and theoretical properties.

B.1 Bi-loop Search Architecture

RAISE's bi-loop structure separates two timescales: the *outer loop* runs continuously (every generation), while the *inner loop* fires periodically (every $\tau = 5$ generations). This separation is motivated by computational efficiency: the inner adversarial search involves re-evaluating an inner population of 8 candidate encodings over the current best heuristic; amortizing this cost over τ outer generations keeps the overhead below 20% of total compute.

At each inner loop invocation, the best heuristic h_t^* from the current outer population is used as the fixed "victim" for the adversarial search. The search seeks low $\mathrm{eval}(h_t^*, s')$ over decoded instances $s' = \mathcal{D}(\mathbf{x}; s_i, \omega)$ whose input encoding satisfies $d(\mathbf{x}, E(s_i)) \le \varepsilon$ for some immutable nominal anchor $s_i \in S_0$. Here d is the mean absolute coordinate difference of Section 2.1. The bound concerns the encoding supplied to the decoder, not necessarily the re-encoding of its output. After the search, the selected instance is appended only to $\mathcal{S}$ and the entire outer population is re-scored by mean fitness. The implementation resets the search inputs to the original nominal instances at each refresh;

accumulated instances never become projection anchors. This is hard-instance augmentation, not a per-candidate minimax evaluation.

B.2 Task-Specific Encodings and Decoding

The field named `distribution` in the implementation stores a bounded feature vector, not a probability histogram. There are no histogram bin edges, and coordinates are not normalized to sum to one. Table 6 summarizes the task-specific meanings. The encoded dimension depends on the task and nominal configuration; it is distinct from the low-dimensional gene vector used to construct perturbations.

Table 6: Feature encodings constrained by the inner search. $D_{\max}$ is the instance's maximum demand parameter.

Task	Encoded coordinates	Nominal dimension
OBP	Item size a_i/C, in arrival order	5,000
Online JSP / offline JSS	Operation duration $(p-1)/98$, grouped by machine in the adapter's slot order	$MJ = 200$
Online VRP	Customer demand $d_i/D_{\max}$, in customer order	50
Offline TSP–GLS	Min–max scaled cluster count, cluster standard deviation, and outlier fraction	3

For TSP, the unscaled parameter ranges are $[1, 5]$, $[0.05, 0.20]$, and $[0, 0.30]$, respectively. These coordinates describe the spatial generator, not the realized empirical distribution of city locations. For equal-length encodings, $d(\mathbf{x}, \mathbf{y}) = n^{-1} \sum_j |x_j - y_j|$ compares coordinates without sorting; its upper bound on $[0, 1]^n$ is 1, not $2/n$. The reported OBP radius sweep uses equal-length vectors of 5,000 items. Optional length-changing paths require alignment of unequal-length vectors and are not used in that sweep; the fixed-dimensional projection argument below does not by itself establish feasibility for such paths.

Decoding and the scope of feasibility. OBP decoding directly sets $a'_i = Cx_i$ at the anchor's capacity. It neither draws items from probabilities nor rounds the resulting values to integers; adversarial training vectors can therefore yield real-valued items, including zeros, although nominal and test data use positive integer sizes. JSP/JSS adapters reconstruct integer operation durations; VRP adapters scale, round, and clip demands; TSP decoding rounds the cluster count and regenerates city coordinates using the specified spatial parameters and seed. These are task-specific transformations, not sampling from a shared probability-mass vector.

The generic radius check is performed on the candidate encoding before these transformations. It is not a universal post-decoding check of $d(E(s'), E(s_i)) \le \varepsilon$. Rounding, operation assignment, or spatial regeneration can change a realized representation. Accordingly, the decoded family $\mathcal{B}_\varepsilon(S_0)$ in the main paper is defined by admissible decoder inputs; we do not assert that all realized instances or their empirical histograms satisfy a common post-decoding radius bound. Establishing such a bound would require a task-specific preservation proof or a decode–re-encode feasibility check.

Fixed-anchor audit (OBP). We audited the saved logs for all twelve runs underlying the reported OBP radius sweep (three per radius). In each run, the five anchor vectors were identical across every refresh and matched the first five entries of the final evaluation set. Table 7 reports $\delta(\mathbf{x}) = \min_{s \in S_0} d(\mathbf{x}, E(s))$ against these original anchors, pooled over logged post-projection candidate records and, separately, all added instances. Candidate records can include repeated evaluations; they are not independent samples. Across 243 refreshes, no logged candidate or added instance exceeded its radius by more than 10^{-9}. This audit is specific to fixed-length OBP, whose direct rescaling preserves the encoding; it does not establish post-decoding feasibility for other tasks.

B.3 Inner Adversarial Instance Search

The inner evolutionary algorithm searches for encoded candidates whose decoded instances degrade the performance of the current best heuristic h^*, subject to the encoded-space radius constraint.

Table 7: Distance to the original nominal anchors in the saved OBP runs. Counts and distances are pooled across three runs per radius. No violations occur at tolerance 10^{-9}.

	Logged candidate encodings			Added instances		
ε	Count	Mean δ	Max δ	Count	Mean δ	Max δ
0.02	2,520	0.02000	0.02000	63	0.02000	0.02000
0.05	2,400	0.05000	0.05000	60	0.05000	0.05000
0.10	2,400	0.09932	0.10000	60	0.09997	0.10000
0.20	2,400	0.17705	0.20000	60	0.19621	0.20000

The reported search operates on an 18-dimensional gene space that parameterizes signed combinations of perturbation functions. Each gene vector produces a candidate encoding; projection leaves an interior candidate unchanged and moves an exterior candidate to the radius boundary. The candidate is checked in encoded space and then decoded by the task adapter. The population is improved over G_{in} generations using selection, crossover, and mutation. Algorithm 2 summarizes the fixed-dimensional procedure; it does not include a post-decoding radius test.

Algorithm 2 Inner Adversarial Instance Search

Require: Current best heuristic h^*, immutable anchors $S_0 = \{s_1, \ldots, s_k\}$, radius ε, inner population size P_{in}, inner generations G_{in}
Ensure: A decoded hard instance from an admissible encoding, if evaluation succeeds
1: Initialize gene population $\mathcal{G} = \{\mathbf{g}_1, \ldots, \mathbf{g}_{P_{\text{in}}}\}$ with $\mathbf{g}_i \sim \text{Uniform}([0,1]^{18})$
2: **for** $g = 1$ to G_{in} **do**
3: **for** each $\mathbf{g}_i \in \mathcal{G}$ **do**
4: $\mathbf{x}_{\text{adv}} \leftarrow \text{construct_encoding}(\mathbf{g}_i)$ ▷ Eq. (3) in main paper
5: $j^* \leftarrow \arg\min_j d(\mathbf{x}_{\text{adv}}, E(s_j))$; $b \leftarrow E(s_{j^*})$
6: $\mathbf{x}_{\text{proj}} \leftarrow b + \frac{\varepsilon}{\max(\varepsilon,\, d(\mathbf{x}_{\text{adv}}, b))}(\mathbf{x}_{\text{adv}} - b)$ ▷ $\varepsilon > 0$
7: Reject candidates that fail the encoded-space radius check
8: $s_i' \leftarrow \mathcal{D}(\mathbf{x}_{\text{proj}}; s_{j^*}, \omega)$ ▷ Task-specific decoding
9: $\text{score}_i \leftarrow -\text{eval}(h^*, s_i')$ ▷ Adversarial score: lower eval = higher adversarial score
10: **end for**
11: Sort $\mathcal{G}$ by score descending; keep top $\lfloor P_{\text{in}}/2 \rfloor$
12: **Crossover**: for each pair of selected parents $(\mathbf{g}_i, \mathbf{g}_j)$, create offspring via uniform crossover with $p_{\text{cross}} = 0.5$
13: **Mutation**: apply Gaussian noise $\mathcal{N}(0, 0.12^2)$ to each gene with $p_{\text{mut}} = 0.35$; clip to $[0,1]^{18}$
14: **end for**
15: **return** The worst successfully evaluated decoded instance from the final generation

Step 1: Population initialization. The inner search begins by randomly initializing $P_{\text{in}} = 8$ gene vectors, each drawn independently from $\text{Uniform}([0,1]^{18})$. This uniform initialization ensures broad coverage of the gene space at the start of the search, giving the evolutionary process diverse starting points without imposing any prior bias toward a particular distributional shift.

Step 2: Candidate-encoding construction. The first nine genes define signed weights $w_i = g_i - 0.5 \in [-0.5, 0.5]$; genes $g_{10:17}$ control the shapes of deterministic perturbation functions. Their weighted sum is added to the mean nominal encoding and clipped coordinatewise to $[0,1]$ (Eq. (3)). This is an additive feature perturbation, not a probability mixture. The optional length gene g_{18} is inactive when the encoded dimension is fixed, as in the reported OBP sweep. No sum-to-one normalization or special probability fallback for an all-zero vector is applied; task-domain validity is separate from box membership.

Step 3: Nearest nominal encoding and radial projection. Let $b \in E(S_0)$ be the nearest original nominal encoding under mean absolute coordinate difference; S_0 is unchanged across refreshes. For $\varepsilon > 0$, the fixed-dimensional projection is

$$\mathbf{x}_{\text{proj}} = b + \alpha(\mathbf{x}_{\text{adv}} - b), \qquad \alpha = \frac{\varepsilon}{\max(\varepsilon, d(\mathbf{x}_{\text{adv}}, b))}.$$

Since $0 \leq \alpha \leq 1$ and both endpoints lie in $[0,1]^n$, the result remains in the box. Homogeneity gives $d(\mathbf{x}_{\text{proj}}, b) = \alpha d(\mathbf{x}_{\text{adv}}, b) \leq \varepsilon$. Exterior candidates reach the boundary; interior candidates are unchanged. This preserves the largest admissible displacement along the chosen direction, without implying that the hardest instance must lie on the boundary. The search checks the projected encoding before evaluation. The argument establishes encoded-space feasibility only; no subsequent probability normalization is performed.

Step 4: Task-specific decoding and adversarial scoring. The adapter constructs $s'_i = \mathcal{D}(\mathbf{x}_{\text{proj}}; s_{j^*}, \omega)$ using the nominal anchor and any required generator seed. OBP directly rescales the ordered item vector; other tasks use the transformations in Appendix B.2. The black-box evaluator computes $\text{eval}(h^*, s'_i)$ and the adversarial score is its negation. Failed evaluations are excluded from successful candidates. The generic search does not check the empirical histogram of a sampled instance or re-encode the decoded instance to enforce a second radius constraint.

Step 5: Selection. After scoring all P_{in} candidates in a generation, the population is ranked by adversarial score in descending order and the top half (i.e., $\lfloor P_{\text{in}}/2 \rfloor = 4$ individuals) is retained as parents for the next generation. This truncation-selection scheme is computationally simple and well-suited to the short inner search horizon. The selected parents define the gene pool from which the next generation is produced via crossover and mutation.

Step 6: Crossover. Each pair of selected parents $(\mathbf{g}_i, \mathbf{g}_j)$ produces one offspring via uniform crossover with probability $p_{\text{cross}} = 0.5$: for each of the 18 gene positions independently, the offspring inherits the value from $\mathbf{g}_i$ with probability 0.5 and from $\mathbf{g}_j$ with probability 0.5. Uniform crossover is chosen over single-point or two-point crossover because it treats all gene positions symmetrically, which is appropriate here given that the 18 genes encode heterogeneous quantities (perturbation weights, shape parameters, and a length parameter) with no meaningful positional ordering.

Step 7: Mutation. Each gene of every offspring is independently perturbed by additive Gaussian noise $\mathcal{N}(0, 0.12^2)$ with probability $p_{\text{mut}} = 0.35$, and the result is clipped to $[0,1]^{18}$ to maintain valid gene values. The mutation standard deviation $\sigma = 0.12$ is chosen to produce moderate perturbations relative to the unit gene range: before clipping, the expected absolute change per gene is $0.35 \times 0.12\sqrt{2/\pi} \approx 0.034$ units, which translates to a meaningful but not catastrophic change in the candidate encoding.

Step 8: Return the selected decoded instance. After G_{in} generations, the worst candidate instance with the highest adversarial score in the final population is returned to the outer loop. It is appended to the instance set $\mathcal{S}$, after which all heuristics in the outer population are re-scored on the updated $\mathcal{S}$ before evolution continues.

Gene decoding detail. The 18-dimensional gene vector $\mathbf{g}$ is partitioned as:

- $g_{1:9}$: basis function weights $w_i = g_i - 0.5 \in [-0.5, 0.5]$
- $g_{10:13}$: powers for Small, Large, Center, and Bimodal
- $g_{14:17}$: Gaussian center, periodic frequency, Poisson-like exponent, and peak position
- g_{18}: optional length gene, inactive in the reported fixed-length OBP sweep

The functions are evaluated deterministically at $q_j = (j-1)/(n-1)$, $j = 1, \ldots, n$, for $n > 1$. Let $z = 2q - 1$ and $p_k = \max(1.01, 1 + 4g_{9+k})$, $k = 1, \ldots, 4$. The historical names describe curve shapes, not probability laws:

1. **Uniform**: $\phi(q) = q$.
2. **Small**: $\phi(q) = q^{p_1}$.
3. **Large**: $\phi(q) = 1 - (1-q)^{p_2}$.
4. **Center**: $\phi(q) = \frac{1}{2} + \frac{1}{2}\,\text{sgn}(z)|z|^{p_3}$.
5. **Bimodal**: $\phi(q) = \frac{1}{2} + \frac{1}{2}\,\text{sgn}(z)|z|^{1/p_4}$.
6. **Gaussian**: $\phi(q) = \exp(-(q - g_{14})^2/(2 \times 0.15^2))$.
7. **Periodic**: $\phi(q) = \frac{1}{2}[1 + \sin(2\pi(1 + 10g_{15})q)]$.

8. **Poisson-like**: $f(q) = (10q)^{\lambda} \exp(-10q)$, $\lambda = 0.1 + 10g_{16}$, and $\phi(q_j) = f(q_j)/(\max_k f(q_k) + 10^{-9})$.
9. **Peak**: $\phi(q) = \exp(-|q - g_{17}|/0.02)$.

B.4 LLM Evolutionary Operators

The outer loop employs five LLM operators from Liu et al. (2024) with distinct roles, as described in Section 2: CREATE, E1, E2, M1, and M2. Table 8 summarizes their inputs, outputs, and invocation characteristics. All operators require the model to first describe the proposed heuristic in one sentence (enclosed in a boxed environment) before implementing the Python function, ensuring interpretable outputs alongside executable code.

Table 8: Summary of RAISE's five LLM evolutionary operators.

Operator	Inputs	Purpose	Parents
CREATE	Task description only	Generate novel heuristic from scratch	0
E1	k parents + task	Generate heuristic with a totally different form from all parents	k
E2	k parents + task	Extract backbone idea from parents; synthesize a new motivated variant	k
M1	One parent + task	Produce a modified version of the parent heuristic	1
M2	One parent + task	Perturb numerical parameters of the parent's scoring function	1

The operators divide into two families. The *exploration* operators (CREATE, E1, E2) are designed to introduce structural novelty: CREATE requires no parent at all; E1 explicitly instructs the model to produce a form *totally different* from all provided parents; and E2 first asks the model to identify the common backbone idea across parents before synthesizing a new heuristic motivated by—but structurally distinct from—that backbone.

The *exploitation* operators (M1, M2) make incremental changes to a single parent. M1 allows free modification of the heuristic's logic, while M2 targets only the numerical parameter settings of the existing scoring function, enabling fine-grained local search without altering algorithmic structure.

CREATE Prompt Template

{task_description}

Create a new heuristic.

1. First, describe your heuristic in one sentence inside boxed {}.
2. Next, implement the following Python function: {function_signature}

Do not give additional explanations.

E1 Prompt Template

{task_description}

I have {k} existing heuristics with their codes as follows:
{indivs}
Please help me create a new heuristic that has a **totally different form** from the given ones.

1. First, describe your heuristic in one sentence. The description must be inside boxed {}.
2. Next, implement the following Python function: {function_signature}

Do not give additional explanations.

E2 Prompt Template

{task_description}

I have {k} existing heuristics with their codes as follows:
{indivs}
Please help me create a new heuristic that has a totally different form from the given ones but can be **motivated** from them.

1. Firstly, identify the common backbone idea in the provided heuristics.
2. Secondly, based on the backbone idea describe your new heuristic in one sentence. The description must be inside boxed {}.
3. Thirdly, implement the following Python function: {function_signature}

Do not give additional explanations.

M1 Prompt Template

{task_description}

I have one heuristic with its code as follows:
{indivs}
Please create a new heuristic that can be a **modified version** of the given one.

1. First, describe your heuristic in one sentence. The description must be inside boxed {}.
2. Next, implement the following Python function: {function_signature}

Do not give additional explanations.

M2 Prompt Template

{task_description}

I have one heuristic with its code as follows:
{indivs}
Please identify the main heuristic parameters and assist me in creating a new heuristic that has **different parameter settings** of the score function provided.

1. First, describe your heuristic in one sentence. The description must be inside boxed {}.
2. Next, implement the following Python function: {function_signature}

Do not give additional explanations.

C Detailed Experimental Design and Settings

This appendix provides complete documentation of the experimental setup, including benchmark task definitions, dataset construction procedures with exact parameterizations derived from the generation scripts, baseline algorithm configurations, RAISE hyperparameter settings, and the full evaluation protocol.

C.1 Benchmark Tasks and Instance Generation

C.1.1 Online Bin Packing (OBP)

Problem definition. In Online Bin Packing, items of integer size $a_i \in \{1, \ldots, C\}$ arrive one at a time from an online sequence $\sigma = (a_1, a_2, \ldots, a_n)$. Upon arrival, each item must be irrevocably assigned to an open bin (without exceeding the bin capacity C) or to a new bin. The objective is to minimize the total number of bins used. Formally, let B denote the number of bins used and

$L = \lceil \sum_i a_i / C \rceil$ the volume lower bound on the offline optimum. The **waste ratio** is:

$$\text{WasteRatio} = \frac{B - L}{L}.$$

Lower values indicate better performance; a waste ratio of 0 means the online algorithm matches the offline lower bound.

Training instances. RAISE's evolutionary search is seeded with a nominal OBP instance set comprising $I = 5$ independent instances in Weibull distribution (Romera-Paredes et al., 2024; Liu et al., 2024), each with $n = 5{,}000$ items and bin capacity $C = 200$. These nominal instances constitute the initial adversarial instance set $\mathcal{S}_0$, which RAISE augments dynamically during the search. All LLM-based baselines except EoH-S (EoH, MoH, ReEvo, and PartEvo) are trained on this same nominal set. EoH-S follows its published protocol and uses 128 diverse training instances; this difference is stated explicitly because access to shifted training data is part of its method.

Test distribution parameterization. Testing evaluates cross-distribution generalization across five standard families. Item sizes are drawn from the continuous distribution, rounded to the nearest integer ($\lfloor \cdot \rceil$), and clipped to the valid range $[1, C]$. Table 9 gives the exact parameterizations used in the generation script.

Table 9: OBP item-size distribution parameterizations (capacity C). All samples are rounded to integers and clipped to $[1, C]$. The Exponential parameter is a scale.

Distribution	Continuous draw	Parameters	Support
Uniform	$a_i \sim \mathcal{U}(1,\, C)$	—	$[1, C]$
Normal	$a_i \sim \mathcal{N}(\mu,\, \sigma^2)$	$\mu = 0.5C,\ \sigma = 0.2C$	clip to $[1, C]$
Lognormal	$a_i \sim \text{LogNormal}(\mu_\ell,\, \sigma_\ell^2)$	$\mu_\ell = \ln(0.25C),\ \sigma_\ell = 0.6$	clip to $[1, C]$
Exponential	$a_i \sim \text{Exp}(\lambda) + 1$	$\lambda = 0.3C$	clip to $[1, C]$
Triangular	$a_i \sim \text{Tri}(\ell,\, m,\, r)$	$\ell = 1,\ m = 0.35C,\ r = C$	$[1, C]$

Test suite structure. The OBP test suite spans all five distributions, three problem sizes ($n \in \{1{,}000,\ 5{,}000,\ 10{,}000\}$), and four bin capacities ($C \in \{100,\ 200,\ 300,\ 400\}$), yielding $5 \times 3 \times 4 = 60$ dataset files, each containing $I = 5$ independently generated instances. The complete test suite thus covers **300 OBP test instances**. Table 10 summarizes the combinatorial structure.

Table 10: OBP test suite summary. Each (distribution, n, C) combination produces one dataset file with $I = 5$ instances.

Problem size n	Distributions	Capacities C	Files	Instances
1,000	5	$\{100, 200, 300, 400\}$	20	100
5,000	5	$\{100, 200, 300, 400\}$	20	100
10,000	5	$\{100, 200, 300, 400\}$	20	100
Total			**60**	**300**

C.1.2 Online Job Shop Scheduling (JSP)

Problem definition. In Online Job Shop Scheduling (OJSP), J jobs must be processed on M machines. Each job j consists of exactly M operations, where operation (j, m) must be processed on machine m for an integer processing time $p_{jm} \in \{1, \ldots, 99\}$. The machine order within each job is a random permutation of all M machines. Jobs carry a release time r_j (before which they cannot be started) and a due date d_j. Jobs arrive online at their release time; the dispatching heuristic selects among eligible operations as machines become available. The objective is to minimize the makespan $C_{\max} = \max_j C_j$, where C_j denotes the completion time of job j.

The **normalized makespan** divides $C_{\max}$ by the theoretical lower bound $\text{LB} = \max\big(1,\ \max_j(r_j + \sum_m p_{jm}),\ \max_m \sum_j p_{jm}\big)$; lower values indicate better performance, with 1.0 denoting an optimal solution.

Instance structure. Each JSP instance is generated as follows.

1. **Operation durations.** For each job j and machine m, the processing time p_{jm} is drawn from the specified distribution (Table 11), with integer values in $[1,\ 99]$.
2. **Machine order.** Each job's operation sequence is a uniformly random permutation of the M machine indices.
3. **Planning horizon.** $H = 2 \times \max(\max_j \sum_m p_{jm},\ \lceil \sum_{j,m} p_{jm}/M \rceil)$.
4. **Release times.** Release times are drawn with random gaps (probability 0.35 of a zero gap, otherwise a random positive increment), then rescaled to fit within a release window of $0.45\,H$; the resulting times are randomly permuted across jobs.
5. **Due dates.** $d_j = r_j + p_j + \delta_j$, where $p_j = \sum_m p_{jm}$, and δ_j combines a multiplicative slack ($\mathrm{Uniform}(1.35,\ 1.75) \times p_j$), an additive congestion allowance ($0.35 \times \bar{p}$, with $\bar{p}$ the mean job workload), and a random noise term bounded by a congestion buffer.

Training instances. The nominal training set comprises $I = 5$ instances with $M = 10$ machines and $J = 20$ jobs. The operation durations are uniformly sampled.

Test distribution parameterization. Table 11 specifies the operation-duration distributions for all five test families. The parameterizations mirror those used for OBP (normalized to the same $[1,\ 99]$ range), enabling direct comparison of distributional effects across benchmarks.

Table 11: JSP operation-duration distribution parameterizations ($D_{\max} = 99$). All samples are rounded to integers and clipped to $[1,\ 99]$. The Exponential parameter is a scale.

Distribution	Continuous draw	Parameters	Support
Uniform	$p_{jm} \sim \mathcal{U}(1,\ D_{\max})$	—	$[1,\ 99]$
Normal	$p_{jm} \sim \mathcal{N}(\mu,\ \sigma^2)$	$\mu = 0.5D_{\max},\ \sigma = 0.2D_{\max}$	clip to $[1,\ 99]$
Lognormal	$p_{jm} \sim \mathrm{LogNormal}(\mu_\ell,\ \sigma_\ell^2)$	$\mu_\ell = \ln(0.25D_{\max}),\ \sigma_\ell = 0.6$	clip to $[1,\ 99]$
Exponential	$p_{jm} \sim \mathrm{Exp}(\lambda) + 1$	$\lambda = 0.3D_{\max}$	clip to $[1,\ 99]$
Triangular	$p_{jm} \sim \mathrm{Tri}(\ell,\ m,\ r)$	$\ell = 1,\ m = 0.35D_{\max},\ r = D_{\max}$	$[1,\ 99]$

Test suite structure. The JSP test suite spans all five distributions, two machine counts ($M \in \{10, 20\}$), and two job counts ($J \in \{20, 50\}$), yielding $5 \times 2 \times 2 = 20$ dataset files, each with $I = 5$ instances. The complete test suite covers **100 JSP test instances**. Table 12 summarizes the structure.

Table 12: JSP test suite: number of instances per $(M, J, \text{distribution})$ cell. Each cell contains $I = 5$ instances. The final row totals instances across distributions.

Distribution	$M = 10,\ J = 20$	$M = 10,\ J = 50$	$M = 20,\ J = 20$	$M = 20,\ J = 50$
Uniform	5	5	5	5
Normal	5	5	5	5
Lognormal	5	5	5	5
Exponential	5	5	5	5
Triangular	5	5	5	5
Total	25	25	25	25

C.1.3 Online Vehicle Routing (VRP)

Problem definition. In Online Vehicle Routing (OVRP), a fleet of V homogeneous vehicles serves $N_c = 50$ customers from a central depot. Each customer i has an integer demand $d_i \in \{1, \ldots, 10\}$ and a location $(x_i, y_i) \in [0, 1]^2$ sampled from a clustered spatial process. Customers arrive online; upon arrival, the routing heuristic irrevocably assigns each customer to a vehicle's route. All vehicles have a fixed capacity of $Q = 40$ units (for $N_c = 50$). The depot is located at the centroid $(0.5,\ 0.5)$ of the unit square.

The reported metric is the inverse composite score $1/r$, where

$$r = 1.2\,s - 2.8\,u - 0.55\,t - 0.10\,b.$$

Here s and u are the fractions of served and rejected customers, t is total travel divided by the number of customers and the spatial scale, and b is the standard deviation of vehicle loads divided by capacity. Lower $1/r$ is better for the positive scores reported here; it measures the combined service, travel, and load-balance objective.

Instance structure. Each VRP instance is generated as follows.

1. **Customer locations.** Locations are drawn from a Gaussian mixture with $K \in \{2, 3\}$ clusters, where each cluster center is sampled uniformly from $[0.15,\ 0.85]^2$, cluster standard deviation is drawn uniformly from $[0.08,\ 0.18]$, and cluster weights follow a symmetric Dirichlet prior. All coordinates are clipped to $[0, 1]^2$.
2. **Customer demands.** Integer demands d_i are drawn from the specified distribution (Table 13), with $d_i \in [1,\ 10]$.
3. **Vehicle capacity.** Fixed at $Q = 40$ for all instances with $N_c = 50$ customers.

Training instances. The nominal training set comprises $I = 5$ instances with 50 customers. The demands are uniformly sampled.

Test distribution parameterization. Table 13 gives the customer-demand distributions for all five test families. Note that the Lognormal parameterization for VRP uses a slightly different location parameter ($0.28\, D_{\max}$ vs. $0.25\, D_{\max}$ for OBP/JSP) and a reduced scale ($\sigma_\ell = 0.55$ vs. 0.60), reflecting the narrower integer demand range.

Table 13: VRP customer-demand distribution parameterizations ($D_{\max} = 10$). All samples are rounded to integers and clipped to $[1,\ 10]$. The Exponential parameter is a scale.

Distribution	Continuous draw	Parameters	Support
Uniform	$d_i \sim \mathcal{U}(1,\ D_{\max})$	—	$[1,\ 10]$
Normal	$d_i \sim \mathcal{N}(\mu,\ \sigma^2)$	$\mu = 0.5 D_{\max},\ \sigma = 0.2 D_{\max}$	clip to $[1,\ 10]$
Lognormal	$d_i \sim \mathrm{LogNormal}(\mu_\ell,\ \sigma_\ell^2)$	$\mu_\ell = \ln(0.28 D_{\max}),\ \sigma_\ell = 0.55$	clip to $[1,\ 10]$
Exponential	$d_i \sim \mathrm{Exp}(\lambda) + 1$	$\lambda = 0.3 D_{\max}$	clip to $[1,\ 10]$
Triangular	$d_i \sim \mathrm{Tri}(\ell,\ m,\ r)$	$\ell = 1,\ m = 0.35 D_{\max},\ r = D_{\max}$	$[1,\ 10]$

Test suite structure. The VRP test suite spans all five distributions and three fleet sizes ($V \in \{5,\ 10,\ 15\}$), yielding $5 \times 3 = 15$ dataset files, each with $I = 5$ instances. The complete test suite covers **75 VRP test instances**, all with $N_c = 50$ customers.

C.1.4 Offline Job Shop Scheduling (JSS)

Problem definition and heuristic interface. The offline task uses the same job structure and release-time, precedence, and machine-capacity constraints as online JSP (Section C.1.2), but all jobs and operation sequences are available before scheduling begins. The evolved function `solve(jobs, n_machines)` is called once and returns a complete schedule: entry (j, k) is the integer start time of operation k of job j. The evaluator checks release-time feasibility, operation precedence, and the absence of overlapping operations on each machine before computing the makespan. Due dates are available as heuristic features but are not hard constraints or part of the makespan objective.

Metric. We report the mean normalized makespan $C_{\max}/\mathrm{LB}$, where

$$\mathrm{LB} = \max\left(1,\ \max_j\left(r_j + \sum_m p_{jm}\right),\ \max_m \sum_j p_{jm}\right).$$

This is the same effective lower bound as for online JSP: the offline evaluator also computes a release-aware per-machine term, but it is dominated by the job-chain and machine-load bounds above. A ratio of 1 certifies attainment of this lower bound; values above 1 need not measure the gap to the true optimum.

Training instances. The nominal dataset contains five instances with $M = 10$ machines and $J = 20$ jobs. Each job visits every machine exactly once in a random permutation, with operation

durations drawn uniformly from the integers $1, \ldots, 99$. The generator uses the planning horizon, clustered release-time procedure, and due-date construction described in Section C.1.2. RAISE and the comparison without the inner loop use this nominal dataset; adversarial search augments the evaluation set only for RAISE.

Test instances and aggregation. The offline evaluation reuses the online JSP test files, rather than generating a separate test collection. It therefore uses the five processing-time distributions and parameterizations in Table 11, with $M \in \{10, 20\}$ and $J \in \{20, 50\}$. There are 20 dataset files containing five instances each, for **100 offline JSS evaluations per heuristic**. These are the same underlying instances counted in the online JSP suite, not 100 additional generated instances. Each family-level result averages the four size configurations (20 instances); the overall mean averages all five families. The evaluated heuristic and its construction logic are given in Appendix F.4.

C.1.5 Offline Traveling Salesperson Problem with Guided Local Search

Problem definition and heuristic interface. Given all n city coordinates, the objective is a minimum-length closed tour visiting every city exactly once. The solver is classical Guided Local Search (GLS); only its `update_edge_distance` guide is evolved. The guide receives the original distance matrix, the current locally optimized tour, and historical edge-penalization counts, and returns modified distances for perturbation moves. Tour construction and the 2-opt and relocate neighborhoods remain fixed, and incumbent tour costs use the original distances (Appendix F.5).

Nominal instance generation. The saved training set contains five synthetic instances with $n = 100$ cities in $[0, 1]^2$. For each instance, two cluster centers are sampled uniformly from $[0.15, 0.85]^2$; mixture weights follow $\mathrm{Dirichlet}(1, 1)$, and each city is assigned to a cluster according to these weights. Independent Gaussian coordinate offsets have standard deviation 0.12, and coordinates are clipped to $[0, 1]$. The nominal outlier fraction is zero. The generator uses base seed 2024. Adversarial scenarios vary the normalized features *number of clusters*, *cluster standard deviation*, and *uniform-outlier fraction*, whose decoding ranges are $[1, 5]$, $[0.05, 0.20]$, and $[0, 0.30]$, respectively; the number of clusters is rounded to an integer and the city count remains fixed. Training evaluates guides with a 10-second GLS budget and at most 1,000 iterations per instance, using the negative relative gap to the two-nearest-neighbor lower bound $L_{\mathrm{NN}} = \frac{1}{2}\sum_i (d_i^{(1)} + d_i^{(2)})$ as fitness, where $d_i^{(1)}$ and $d_i^{(2)}$ are the two shortest distances from city i to other cities.

Held-out TSPLIB benchmark. Testing uses a curated set of 47 `EUC_2D` TSPLIB instances, not samples from the training generator. The selection window is 50–1,000 cities; the included instances actually range from 51 to 783 cities. The size bands contain six instances with $50 \le n < 100$, 21 with $100 \le n < 200$, and 20 with $200 \le n \le 1{,}000$. The evaluation list excludes `tsp225` because its supplied tour disagrees with the reference value, and `linhp318` because it is a Hamiltonian-path variant. All methods use the same instances, nearest-neighbor initialization, 60-second GLS budget, 1,000-iteration cap, and one guided perturbation phase per iteration. The solver uses seed 2024. The overall result is the mean over all 47 instances, not an unweighted mean of the three size-band averages.

Distance convention and reported gap. The evaluator jointly rescales both coordinate axes using a single range $s = \max_{i,a} x_{ia} - \min_{i,a} x_{ia}$, preserving the geometry, and computes unrounded Euclidean distances. If L is the returned tour length on these scaled coordinates and L_{ref} is the stored TSPLIB reference cost, it reports $100\,(L - L_{\mathrm{ref}}/s)/(L_{\mathrm{ref}}/s)$. The reference costs use TSPLIB's integer-rounded distance convention, whereas the search costs are unrounded. Thus the reported values are deviations from the stored reference under this implementation, rather than exact optimality gaps for a single common distance convention. This test-time reference is distinct from the synthetic training lower bound.

C.2 Baseline Algorithm Descriptions

C.2.1 Classical Baselines for OBP

BestFit. Each arriving item is placed in the open bin with the least remaining capacity that still accommodates the item (tightest-fit first). If no open bin can accommodate the item, a new bin is opened. BestFit is a deterministic algorithm with no hyperparameters and achieves an asymptotic competitive ratio of 1.7.

FirstFit. Each arriving item is placed in the *first* open bin (in order of creation) that can accommodate it, opening a new bin only if no existing bin suffices. FirstFit is also deterministic with asymptotic competitive ratio 1.7, but typically performs slightly worse than BestFit because it does not account for remaining capacity when selecting the bin.

C.2.2 CLASSICAL BASELINES FOR JSP

Shortest Processing Time (SPT). Upon job arrival, operations are scheduled to prioritize the job with the shortest total remaining processing time among all ready operations. SPT tends to minimize average completion time but can sacrifice makespan on large instances.

Minimum Slack (MinSlack). Dispatching priority is assigned to the job with the smallest slack value $s_j = d_j - r_j - \sum_m p_{jm}$, where d_j is the due date and r_j the release time. MinSlack targets due-date adherence but can produce suboptimal makespan when slack values cluster closely.

Earliest Due Date (EDD). Operations are scheduled by ascending due date d_j. EDD is optimal for minimizing maximum lateness in the single-machine case but offers no direct makespan guarantee on multi-machine instances.

C.2.3 CLASSICAL BASELINES FOR VRP

Nearest-Feasible Insertion (NF-Ins.). Each arriving customer is inserted at the cheapest feasible position (minimum route-length increase) in the vehicle route with the highest remaining capacity that can serve the customer's demand. If no feasible vehicle exists, a new route is initialized.

Slack-Preserving Insertion (SP-Ins.). Insertion cost is adjusted to penalize routes that leave little residual capacity, thereby preserving flexibility for future customer arrivals. The customer is assigned to the vehicle whose insertion minimizes the normalized insertion cost relative to remaining capacity slack.

Urgency-Weighted Insertion (UW-Ins.). Insertion cost is weighted by a demand urgency factor, which increases the priority of high-demand customers. The customer is routed to the vehicle that minimizes the urgency-weighted insertion distance.

C.2.4 LLM-BASED BASELINES

All LLM-based methods are initialized with the same task description and function signature, and receive the same total LLM sampling budget ($N_{\max} = 1{,}000$ samples). Training-instance protocols are described in Section C.1, including the diverse set used by EoH-S.

EoH (Liu et al., 2024). Evolution of Heuristics maintains a population of heuristic Python functions and applies an initialization prompt (CREATE) and four evolutionary operators (E1, E2, M1, and M2). Population size is 10; no memory component. The heuristic trained on nominal instances is evaluated directly on all shifted test distributions without any re-training.

MoH (Shi et al., 2026). MoH (Meta-objective Heuristics) employs an LLM to iteratively refine a meta-optimizer that autonomously constructs and diversifies heuristic-optimizers via self-invocation, eliminating reliance on a fixed evolutionary computation template. The constructed heuristic-optimizers then evolve downstream heuristics. MoH further applies a multi-task training scheme across multiple problem instances to promote cross-instance generalization.

ReEvo (Ye et al., 2024). Reflective Evolution augments EoH with a dual-memory architecture: a short-term memory of recent high-scoring programs and a long-term reflective memory generated by the LLM analyzing performance gaps. Memory injection guides subsequent LLM calls toward progressively better regions of heuristic space. Configuration: same population size and budget as EoH, plus memory components enabled.

PartEvo (Hu & Zhang, 2025). Partition-based Evolution incorporates feature-assisted niche construction within an abstract search space, enabling seamless integration of niche-based strategies from evolutionary computation. Niching encourages maintenance of a diverse heuristic population, mitigating premature convergence.

EoH-S (Liu et al., 2026). Evolution of Heuristic Set (EoH-S) introduces portfolio-based AHD: rather than seeking a single best heuristic, it discovers a diverse set of complementary heuristics forming a Pareto-optimal portfolio. EoH-S is particularly relevant as a distribution-shift baseline because maintaining a diverse portfolio implicitly confers some cross-distribution robustness. Results report a single heuristic from the EoH-S portfolio, the export `algorithm_eohs_top3_original.py`, evaluated identically to other methods.

C.3 RAISE Hyperparameter Settings

Table 14 provides a complete listing of all RAISE hyperparameters, including the values used in experiments and the OBP radius sweep.

Table 14: Complete hyperparameter settings.

Parameter	Symbol	Value Used
Outer loop parameters		
Population size	$\mathcal{P}_{max}$	10
Max samples (budget)	$N_{\max}$	1000
Refresh interval	τ	5
Inner search parameters		
Inner population size	P_{in}	8
Inner generations	G_{in}	4
Gene dimension	—	18
Mutation std. dev.	σ	0.12
Mutation probability	p_{mut}	0.35
Crossover probability	p_{cross}	0.5

The OBP headline results and full-model row in the main ablation table use $\varepsilon = 0.2$. The aggregation, adaptive-weighting, and nominal-distribution sensitivity studies retain $\varepsilon = 0.1$ so that all variants within each study remain directly comparable.

D Comprehensive Experimental Results

This appendix reports the complete numerical results for all three benchmark problems across every evaluated distribution and configuration. **Bold** marks the best result in each column.

D.1 Online Bin Packing (OBP)

The metric is the excess-bin ratio (%), i.e., $(B - L)/L \times 100$, where B is the number of bins used and $L = \lceil \sum_i a_i/C \rceil$ is the volume lower bound; lower is better.

Per-size breakdown. Tables 15, 17, and 18 detail results at each problem size, averaged over the four bin capacities.

D.2 Online Job Shop Scheduling (JSP)

The metric is normalized makespan (lower is better). Tables 19 and 20 report results for 10-machine and 20-machine instances respectively. Columns are grouped by distribution; within each group the two job-count variants (20-job and 50-job) are shown side by side. The *Avg* column averages across all ten (distribution, job-count) cells within the table.

Table 15: OBP shifted waste ratio (%) at $n = 1000$, averaged over $C \in \{100, 200, 300, 400\}$. The RAISE row uses $\varepsilon = 0.2$ and averages three runs. Lower is better.

Method	Uniform	Normal	Lognormal	Exponential	Triangular	Avg
EoH	9.97	9.82	3.27	6.36	10.18	7.92
MoH	12.11	13.75	4.23	6.02	13.77	9.97
ReEvo	10.89	11.82	3.23	5.86	9.26	8.21
PartEvo	7.72	8.35	2.12	2.67	7.42	5.66
EoH-S	5.06	4.40	**1.58**	1.01	**2.97**	3.01
RAISE$^{\varepsilon=0.2}$	**4.04**	**3.76**	1.92	**0.62**	3.68	**2.80**

Table 16: OBP waste ratio (%) on n=5000, C=200 instances. RAISE values are means over three independent runs. Best per distribution in **bold**. Underline: second best. Lower is better.

Method	Uniform	Normal	Lognormal	Exponential	Triangular	Avg.
BestFit	1.717	2.803	1.914	**0.300**	3.006	1.948
FirstFit	2.928	4.096	2.049	0.450	3.696	2.644
EoH (Liu et al., 2024)	5.182	6.308	1.321	5.303	5.765	4.776
ReEvo (Ye et al., 2024)	5.466	6.539	**0.714**	3.011	3.218	3.790
PartEvo (Hu & Zhang, 2025)	3.853	5.062	0.835	2.236	4.619	3.321
EoH-S (Liu et al., 2026)	2.318	3.010	1.334	0.436	**2.043**	1.828
MoH (Shi et al., 2026)	5.277	6.531	0.917	1.443	5.113	3.856
RAISE$^{\varepsilon=0.02}$	4.905	4.698	0.773	2.022	3.031	3.086
RAISE$^{\varepsilon=0.05}$	2.126	3.106	1.478	0.382	2.246	1.868
RAISE$^{\varepsilon=0.1}$	**1.702**	2.765	1.959	0.350	2.514	1.858
RAISE$^{\varepsilon=0.2}$	1.789	**2.726**	1.734	0.382	2.213	**1.769**

Table 17: OBP shifted waste ratio (%) at $n = 5000$, averaged over $C \in \{100, 200, 300, 400\}$. The RAISE row uses $\varepsilon = 0.2$ and averages three runs. Lower is better.

Method	Uniform	Normal	Lognormal	Exponential	Triangular	Avg
EoH	5.02	5.88	1.32	5.25	5.81	4.66
MoH	5.72	7.01	1.06	1.68	5.54	4.20
ReEvo	5.80	6.76	**0.89**	3.17	3.46	4.02
PartEvo	4.17	5.33	1.04	2.40	4.73	3.53
EoH-S	2.36	3.00	1.33	0.40	**2.08**	1.83
RAISE$^{\varepsilon=0.2}$	**1.84**	**2.77**	1.72	**0.37**	2.26	**1.79**

Table 18: OBP shifted waste ratio (%) at $n = 10000$, averaged over $C \in \{100, 200, 300, 400\}$. The RAISE row uses $\varepsilon = 0.2$ and averages three runs. Lower is better.

Method	Uniform	Normal	Lognormal	Exponential	Triangular	Avg
EoH	3.94	4.02	0.79	5.00	5.15	3.78
MoH	4.53	5.03	0.52	0.86	4.75	3.14
ReEvo	4.75	4.95	**0.44**	2.54	2.31	3.00
PartEvo	3.32	3.87	0.61	1.90	5.02	2.94
EoH-S	2.02	2.02	1.27	**0.28**	**1.96**	1.51
RAISE$^{\varepsilon=0.2}$	**1.63**	**1.88**	1.64	0.31	2.04	**1.50**

Table 19: JSP normalized makespan on 10-machine instances (lower is better). Two job counts (20j / 50j) per distribution. *Avg* is the mean over all ten columns. **Bold**: best; underline: second best.

	Uniform		Normal		Lognormal		Exponential		Triangular		
Method	20j	50j	20j	50j	20j	50j	20j	50j	20j	50j	**Avg**
SPT	1.3980	1.2690	1.5062	1.2456	1.5293	1.2586	1.3885	1.2723	1.4360	1.2570	1.3561
MinSlack	1.4158	1.2612	1.5065	1.2727	1.4965	1.2961	1.4965	1.3561	1.4203	1.2902	1.3812
EDD	1.5291	1.2904	1.5372	1.3260	1.5347	1.3229	1.5383	1.3736	1.4936	1.2870	1.4233
EoH	1.2456	1.1513	**1.2805**	**1.1918**	1.2952	**1.1957**	1.2905	**1.1689**	1.2912	1.1730	1.2284
ReEvo	1.2618	**1.1486**	1.2940	1.2238	1.3326	1.2205	**1.2750**	1.2006	1.2692	1.1954	1.2421
RAISE	**1.2378**	1.1696	1.2819	1.2135	**1.2748**	1.2143	1.2795	1.1809	**1.2448**	**1.1693**	**1.2266**

Table 20: JSP normalized makespan on 20-machine instances (lower is better). Two job counts (20j / 50j) per distribution. *Avg* is the mean over all ten columns. **Bold**: best; underline: second best.

	Uniform		Normal		Lognormal		Exponential		Triangular		
Method	20j	50j	20j	50j	20j	50j	20j	50j	20j	50j	**Avg**
SPT	1.3117	1.4749	1.3784	1.5482	1.2630	1.4332	1.2948	1.4193	1.3122	1.5199	1.3956
MinSlack	1.3051	1.4961	1.3877	1.5204	1.3671	1.4633	1.3735	1.5327	1.3988	1.5009	1.4346
EDD	1.3278	1.5939	1.3804	1.5948	1.4686	1.5498	1.4024	1.5830	1.4006	1.5844	1.4886
EoH	1.2109	**1.2611**	1.2557	1.3148	1.2486	1.2950	1.2047	1.2517	1.2072	1.2896	1.2539
ReEvo	1.1840	1.3035	1.2412	1.3241	1.2250	1.2815	**1.1813**	**1.2394**	1.2215	1.2989	1.2500
RAISE	**1.1752**	1.2755	**1.2210**	**1.3128**	**1.2019**	**1.2677**	1.1936	1.2676	**1.1896**	**1.2850**	**1.2390**

D.3 Online Vehicle Routing (VRP)

The metric is the inverse composite score $1/r$, with r defined in Appendix C.1.3; lower $1/r$ is better for the positive scores reported here. In two instances (Uniform and Triangular with 15 vehicles) the Nearest-Feasible baseline achieves $r < 1.0$, giving $1/r > 1.0$; these are marked with †. Tables 21–23 report results for each fleet size separately. Column abbreviations: U=Uniform, N=Normal, LN=Lognormal, Exp=Exponential, Tri=Triangular.

Table 21: VRP inverse composite score $1/r$, fleet size 5 vehicles. Lower is better; **bold** marks the best per column.

Method	**U**	**N**	**LN**	**Exp**	**Tri**	**Avg**
NF-Ins.	0.9423	0.9262	0.9248	0.9359	0.9444	0.9347
SP-Ins.	0.9009	0.8930	0.8935	0.8956	0.9025	0.8971
UW-Ins.	0.9060	0.8946	0.8946	0.8993	0.9057	0.9000
EoH	0.9049	0.8990	0.8986	0.9004	0.9055	0.9017
ReEvo	0.9034	**0.8902**	0.8921	0.8953	0.9022	0.8966
RAISE	**0.8975**	**0.8902**	**0.8889**	**0.8918**	**0.8970**	**0.8931**

E Extended Discussion and Additional Studies

This appendix expands the discussion of Section 5 with the complete experimental record behind the summary numbers quoted in the main text. It covers the nominal–robustness trade-off and the comparison against a specialized online policy (Appendix E.1), the choice of aggregation and instance weighting in the outer loop (Appendix E.2), the offline tasks (Appendix E.3), sensitivity to the nominal training distribution (Appendix E.5), and the scope of the search-space prior together with the routes that would remove it (Appendix E.6).

Table 22: VRP inverse composite score $1/r$, fleet size 10 vehicles. Lower is better; **bold** marks the best per column.

Method	U	N	LN	Exp	Tri	Avg
NF-Ins.	0.9895	0.9677	0.9639	0.9668	0.9859	0.9748
SP-Ins.	0.9398	0.9285	0.9417	0.9309	0.9389	0.9360
UW-Ins.	0.9440	0.9285	0.9283	0.9292	0.9428	0.9346
EoH	**0.9297**	0.9235	0.9230	0.9267	0.9337	0.9273
ReEvo	0.9321	**0.9198**	0.9341	0.9262	**0.9312**	0.9287
RAISE	0.9345	0.9228	**0.9215**	**0.9212**	0.9316	**0.9263**

Table 23: VRP inverse composite score $1/r$, fleet size 15 vehicles. Lower is better; **bold** marks the best per column. † marks entries where the raw composite score $r < 1$.

Method	U	N	LN	Exp	Tri	Avg
NF-Ins.	$1.0180^{\dagger}$	0.9931	0.9877	0.9904	$1.0249^{\dagger}$	1.0028
SP-Ins.	0.9745	0.9553	0.9732	0.9562	0.9734	0.9665
UW-Ins.	0.9725	0.9537	0.9712	0.9639	0.9736	0.9670
EoH	**0.9637**	0.9529	**0.9496**	0.9494	0.9651	**0.9561**
ReEvo	0.9668	0.9631	0.9618	0.9516	**0.9635**	0.9613
RAISE	0.9709	**0.9508**	0.9499	**0.9487**	0.9646	0.9570

E.1 Nominal Performance and Comparison with a Specialized Online Policy

E.1.1 Why nominal performance is reported separately

RAISE selects heuristics by mean performance on nominal instances and discovered hard instances from $\mathcal{B}_\varepsilon(S_0)$, rather than on the nominal set alone. Reporting only shifted performance would therefore hide the quantity a practitioner most needs in order to decide whether to adopt the method: how much nominal performance is being spent. Table 24 reports both regimes for every method at the n=5000, C=200 configuration, with the nominal column measured on the five Weibull training instances. The DRL policy was trained on the same nominal distribution.

Table 24: Nominal versus shifted performance on online bin packing (n=5000, C=200), excess-bin ratio (%, lower is better). *Nominal*: the five Weibull training instances. *Shifted*: mean over the five OOD distribution families (Uniform, Normal, Lognormal, Exponential, Triangular). RAISE entries average three independent runs. *Degradation* is the ratio shifted/nominal; values close to $1\times$ indicate a method whose behaviour transfers.

Category	Method	Nominal	Avg. shifted	Degradation
LLM-based AHD	EoH	0.199	4.776	$24.0\times$
	ReEvo	**0.139**	3.790	$27.3\times$
	PartEvo	0.199	3.321	$16.7\times$
	MoH	0.278	3.856	$13.9\times$
Specialized online DRL	DRL (checkpoint 500)	0.951	2.607	$2.7\times$
	DRL (checkpoint 800)	0.753	2.383	$3.2\times$
	DRL (checkpoint 1,000)	0.614	2.394	$3.9\times$
	DRL (checkpoint 1,200)	0.634	2.105	$3.3\times$
RAISE (ours)	$\varepsilon = 0.02$	0.484	3.086	$6.4\times$
	$\varepsilon = 0.05$	1.345	1.868	$1.4\times$
	$\varepsilon = 0.1$	1.670	1.858	$\mathbf{1.1}\times$
	$\varepsilon = 0.2$	1.485	**1.769**	$1.2\times$

Three conclusions follow, and we state them without softening.

RAISE is worse than nominal-optimized AHD on the nominal distribution. At the primary $\varepsilon = 0.2$ setting, RAISE attains 1.485% against 0.139–0.199% for ReEvo, EoH, and PartEvo. This is approximately the level of classical BestFit (1.551%). The earlier formulation "RAISE remains competitive on nominal instances" overstated the evidence and has been replaced throughout by the explicit trade-off statement.

RAISE transfers much more evenly across regimes. At $\varepsilon = 0.2$ its degradation ratio is $1.2\times$, against 13.9–$27.3\times$ for the nominal-optimized LLM-based methods and 2.7–$3.9\times$ for the DRL checkpoints. Where the deployment distribution is known, a nominal-optimized method is the better choice. RAISE addresses the case where it is not, and where the cost of being wrong is an order of magnitude.

ε controls the empirical nominal–robustness trade-off. At $\varepsilon = 0.02$, nominal waste falls to 0.484% but shifted waste rises to 3.086%. The larger radii give substantially lower shifted waste, with $\varepsilon = 0.2$ attaining the best shifted average (1.769%). The finite-run values are not perfectly monotone—for example, $\varepsilon = 0.2$ also has lower nominal waste than $\varepsilon = 0.1$—so the sweep should be read as an empirical trade-off under stochastic search rather than a deterministic calibration curve. As $\varepsilon \to 0$, the objective itself reduces to standard AHD.

E.1.2 The specialized online DRL baseline

Reviewers of an earlier version of this work asked for a baseline designed specifically for online decision-making rather than a general-purpose AHD framework. To the best of our knowledge there is currently no LLM-driven AHD method proposed specifically for online optimization. We did identify several task-specific deep-learning approaches that learn online decision policies directly, including constrained deep reinforcement learning for online 3D bin packing (Zhao et al., 2021), neural combinatorial optimization for online vehicle routing (James et al., 2019), and spatiotemporal graph neural networks for online distributed job-shop scheduling (Liu et al., 2025).

These methods are not LLM-based AHD: they represent the decision rule as a neural policy rather than an explicit heuristic program, and they are trained by gradient descent on freshly sampled instances rather than by program search over a fixed instance set. They nevertheless provide the most informative available comparison for the question that was raised, namely whether the robustness advantage of RAISE survives contact with a method specialized for the task.

Setup. Among the identified works, Zhao et al. (2021) was the one with an official public implementation that could be reliably adapted. We adapted it to the 1D online bin-packing setting of this paper, trained it on the same Weibull nominal distribution, and evaluated it on exactly the same nominal and five shifted test distributions used for all other methods. The learning and optimization procedures remained method-specific. We report *all* evaluated checkpoints rather than selecting the most favourable one, so that the comparison does not depend on checkpoint choice.

Results. The DRL policy is a substantially stronger robustness baseline than the EoH, ReEvo, PartEvo, and MoH: every reported checkpoint attains a better shifted average (2.105–2.607%) than these four baselines (3.321–4.776%). EoH-S, at 1.828%, remains better than all reported DRL checkpoints. RAISE nevertheless attains the lowest shifted average of all evaluated methods, 1.769% against 2.105% for the best checkpoint, and outperforms every individual checkpoint. At the same time the results expose a nominal-versus-robustness trade-off rather than uniform dominance: ReEvo achieves the best nominal performance, the DRL policy occupies an intermediate position in both regimes, and RAISE achieves the best performance under shift.

E.2 Aggregation and Instance Weighting in the Outer Loop

Section 2.3 scores heuristics by the weighted aggregate

$$\hat{r}_\lambda(h) \;=\; \sum_{k=1}^{K} \lambda_k \,\mathrm{eval}(h, s'_k), \qquad \lambda \in \Delta^{K-1},$$

and uses the uniform weighting $\lambda_k = 1/K$. This subsection reports the evidence for that choice against the two natural alternatives inside the same family (worst case and CVaR) and against two adaptive weighting schemes.

E.2.1 AGGREGATION FUNCTION: MEAN, WORST CASE, AND CVAR

Table 25: Effect of the aggregation function on online bin packing at $\varepsilon = 0.1$. Mean excess-bin ratio (%, lower is better) under shifted distributions, averaged over all n, all capacities, and three random seeds. All other components of RAISE are held fixed. **Bold**: best per column.

Aggregation	Uniform	Normal	Lognormal	Exponential	Triangular	**Avg.**
Worst case (min)	3.225	3.453	1.162	0.823	3.836	2.500
CVaR	3.405	3.685	**1.048**	0.777	4.680	2.719
Mean (used in RAISE)	**2.303**	**2.665**	2.015	**0.475**	**2.781**	**2.048**

Table 25 shows that uniform aggregation attains the best average (2.048%) and is best on four of the five distribution families, ahead of worst-case aggregation (2.500%, $+22\%$ relative) and CVaR (2.719%, $+33\%$ relative).

A possible explanation is that hard-instance augmentation exposes weaknesses while mean aggregation moderates the influence of individual instances. Under worst-case aggregation, fitness depends entirely on the single hardest instance in $\mathcal{S}$, whose identity can change after a refresh. CVaR averages a small tail, whereas uniform aggregation spreads weight over the entire accumulated set. This may stabilize selection, but the table measures final shifted performance, not selection-signal variance or this mechanism directly.

Equation (1) motivates searching for high-loss instances, but RAISE attacks only the current best heuristic and optimizes mean fitness over the accumulated set. The result here favors mean aggregation over finite-set worst-case selection on the average-shift metric in this OBP setting at $\varepsilon = 0.1$. It neither compares against a solver of the full minimax problem nor establishes worst-case robustness over $\mathcal{B}_\varepsilon(S_0)$.

E.2.2 ADAPTIVE INSTANCE WEIGHTING: HEDGE AND AGE DECAY

Uniform weighting implies that the influence of each newly discovered adversarial instance decreases as $1/K$ while $\mathcal{S}$ grows. A natural question is whether learning λ during the search recovers the influence of recent discoveries. We evaluate two schemes.

Hedge. Weights are updated online by exponentiated gradient. At refresh t,

$$\lambda_k^{(t+1)} \;\propto\; \lambda_k^{(t)} \exp\!\big(\eta\, \ell_k^{(t)}\big),$$

where $\ell_k^{(t)}$ is the per-instance loss of the current population on instance k, normalized across instances, and the result is renormalized onto the simplex. A newly added adversarial instance enters with $\lambda_k = 1/K$. We sweep $\eta \in \{0.5,\, 1,\, 2,\, 4\}$. Larger η concentrates weight faster on instances that remain difficult.

Age decay. Weights decay exponentially with instance age,

$$\lambda_k \;\propto\; \gamma^{\,t-t_k},$$

normalized over $\mathcal{S}$, where t_k is the refresh at which instance k was added and t is the current refresh. We sweep $\gamma \in \{0.5,\, 0.7,\, 0.9\}$. The newest instance approaches weight $1-\gamma$ as the history grows instead of decaying to $1/K$ as under uniform averaging.

Compute budget. Each adaptive-weighting configuration was evaluated with a single run in order to maximize coverage of the hyperparameter space. To estimate the noise floor, the best-performing Hedge configuration ($\eta = 0.5$) was additionally repeated with three independent runs.

Table 26 supports three statements.

Mild adaptive weighting gives at most a modest improvement. Hedge with $\eta = 0.5$ improves nominal performance from 1.683 to 1.360 while also achieving the second-best shifted result (2.019 against 2.048). Age decay with $\gamma = 0.7$ achieves the best shifted average (2.000). Hedge grants newly discovered adversarial instances more influence than uniform averaging, so they affect selection immediately rather than being diluted by older instances, while also emphasizing instances that

Table 26: Adaptive instance weighting on online bin packing at $\varepsilon = 0.1$, excess-bin ratio (%, lower is better). *Avg. shifted* is averaged over the five shifted item-size distributions. **Bold**: best per column; underline: second best.

Weighting	Nominal	Avg. shifted
Uniform mean (RAISE default)	1.683	2.048
Hedge $\eta = 0.5$	**1.360**	<u>2.019</u>
Hedge $\eta = 1$	<u>1.466</u>	2.178
Hedge $\eta = 2$	1.743	2.036
Hedge $\eta = 4$	1.901	2.644
Age decay $\gamma = 0.5$	23.870	4.909
Age decay $\gamma = 0.7$	1.704	**2.000**
Age decay $\gamma = 0.9$	1.822	2.092

remain difficult for the current population. The margin over uniform weighting, however, is 0.05 percentage points.

Weight concentration is actively harmful. Age decay with $\gamma = 0.5$ assigns nearly half of the total weight to the newest adversarial instance. Shifted performance deteriorates from 2.048 to 4.909 and nominal performance collapses to 23.870. When a few instances dominate the objective, the optimization continually reorients toward whichever adversarial instance was most recently emphasized and never converges. This is the same failure mode observed for worst-case aggregation, which is the extreme point of the same axis.

The uniform mean remains the right default. It is parameter-free, cannot enter the concentrated regime, and performs within 0.05 points of the best swept configuration. Choosing between the adaptive configurations would require shifted validation data, which is precisely what the setting assumes is unavailable. We therefore keep uniform weighting in RAISE and treat adaptive weighting under explicit effective-sample-size control as future work.

E.2.3 Growth of the adversarial instance set

The instance set grows monotonically and is never pruned, but it is bounded by the sample budget. With $\tau = 5$ and 200 outer generations there are 40 refreshes, so for OBP $|\mathcal{S}|$ grows from the 5 nominal instances to 45. Growth is linear in the number of generations and terminates when the LLM sample budget is exhausted.

The nominal weight of each instance therefore decreases as $1/K$, but this does not characterize its influence. Whenever a new adversarial instance is appended, the entire population is re-scored (Algorithm 1, line 13) and survival rankings can change; an instance with small individual weight can still eliminate heuristics that fail on it. Since instances are never discarded, every historical adversarial example continues to constrain the search for the remainder of the run. The corresponding number of heuristic–instance evaluations is approximately $17{,}880$ for OBP, more than EoH or ReEvo ($\approx 5{,}000$) and far below EoH-S ($\approx 128{,}000$); see Appendix G.

E.3 Offline Tasks: Job-Shop Scheduling and TSP on TSPLIB

The inner loop requires only a black-box evaluation function. It therefore does not depend on the decision rule being consumed online, on the solver being learned, or on the instances being synthetic. We test all three of these independently.

E.3.1 Offline job-shop scheduling

Setup. The task is classical offline JSS: all jobs and their operation sequences are known in advance, and the heuristic under evolution is the dispatching rule used to construct the schedule. The reported metric is the makespan ratio against the evaluator's lower bound (lower is better), averaged over all problem configurations for each of the five processing-time distribution families. The comparison is against the identical pipeline with the inner loop removed, which is standard LLM-driven AHD on

the nominal set; this isolates the contribution of constrained adversarial instance search rather than that of LLM-based program search.

Table 27: Offline job-shop scheduling: mean makespan ratio (lower is better), averaged over all problem configurations for each distribution family. **Bold**: best per column.

Method	Uniform	Normal	Lognormal	Exponential	Triangular	**Avg.**	**Worst**
RAISE w/o inner loop	1.2507	1.2660	1.2620	1.2480	1.2412	1.2536	1.2660
RAISE	**1.1939**	**1.2315**	**1.2254**	**1.2031**	**1.2107**	**1.2129**	**1.2315**

Result. Table 27 shows that adding the inner loop improves the makespan ratio on all five distribution families, by 0.041 on average and by 0.035 in the worst case. The improvement is uniform rather than concentrated in a single family, which is the signature expected of a mechanism that broadens the evaluation set rather than one that specializes to a particular shift.

E.3.2 Offline TSP with Guided Local Search on TSPLIB

Setup. Here the solver is a classical, non-learned Guided Local Search (Voudouris et al., 2018), and RAISE evolves only the guide heuristic that determines which edges are penalized. Neither the local search nor the solution construction is learned. Evaluation uses 47 real TSPLIB instances (Reinelt, 1991) with between 50 and 1000 cities, grouped into three size bands. The reported metric is the mean optimality gap (%) against the known optimal tours.

This is the most demanding of our transfer tests for one specific reason: TSPLIB instances are drawn from real geographic and circuit-layout data and were never sampled from any generator used at training time. The distribution shift is real rather than synthetic, and neither the nominal set nor the nine perturbation functions were designed with TSPLIB geometry in mind.

Table 28: Offline TSP on real TSPLIB instances: mean optimality gap (%, lower is better) of GLS-TSP, grouped by number of cities (47 instances in total). **Bold**: best per column.

Method	50–100 (n=6)	100–200 (n=21)	200–1000 (n=20)	**Avg.**
GLS (classical)	2.9920	3.4281	3.3906	3.3565
RAISE w/o inner loop	0.5336	0.1787	0.9323	0.5447
RAISE	**0.4848**	**0.1602**	**0.8454**	**0.4932**

Result. Table 28 shows that the mean optimality gap improves from 0.5447% to 0.4932%, with the better result in every size band, against 3.3565% for classical GLS. The absolute margin is small because the LLM-evolved guide heuristic is already strong; the relevant observation is that the improvement is consistent across bands on real benchmark instances, which supports the claim that the mechanism is not restricted to online problems, learned solvers, or generated instances.

E.3.3 Why we do not evaluate on neural constructive solvers

A natural further question is whether the inner loop also robustifies neural constructive solvers. We argue this is not the most informative probe for the mechanism we claim. Neural constructive solvers optimize weights by gradient descent and are typically trained on 10^5–10^6 freshly sampled instances, so "the training instance set" is not a scarce fixed object for them, and their robustification is a well-studied and largely separate problem (Zhou et al., 2023). Placing an ε-ball inner loop on top of a solver that resamples its training distribution every epoch would not isolate the effect we are measuring. The setting RAISE addresses is the one where the training instance set is small, fixed, and the only evidence about the deployment distribution that is available.

E.4 Convergence Behaviour of the Bi-Level Search

Figure 3 illustrates the convergence dynamics of RAISE on OBP, showing the heuristic score (larger is better; a negative sign is used for illustration) as a function of the number of populations. The left panel presents the evolution of both the best heuristic score and the population mean over generations,

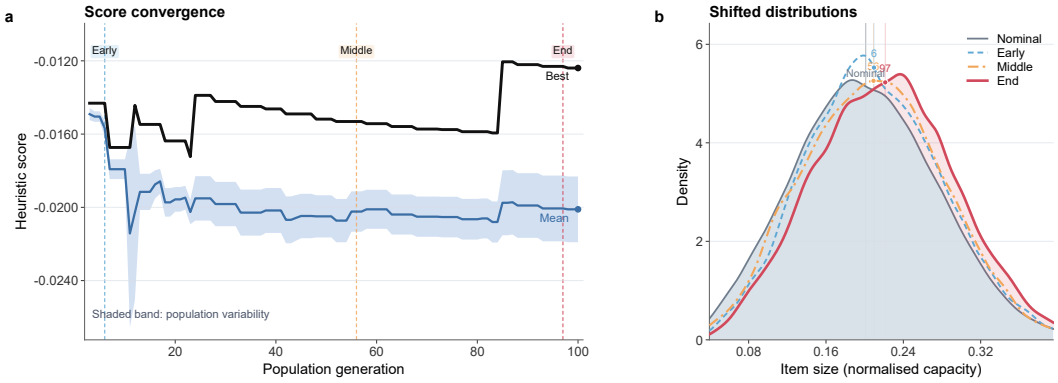


Figure 3: Robust adversary search convergence in online bin packing. **Left:** population score evolution across generations. **Right:** adversarial distribution shift over search stages.

revealing clear stage-wise behavior. Following an initial exploration phase, the population rapidly adapts to increasingly challenging evaluation instances, indicating progressive consolidation of robust design principles rather than isolated lucky discoveries. The ramp-up stage corresponds to the emergence of a new robust heuristic, while the decreasing stage reflects the exploration of new worst-case instances, during which the current heuristic score declines.

The right panel further reveals that this convergence is accompanied by a systematic shift in the adversarial item-size distribution away from the nominal regime toward harder distributions that expose the weakness of conventional packing rules. Taken together, these trends show that RAISE does not simply optimize for nominal performance, but converges by continually identifying, attacking, and repairing distribution-sensitive failure modes.

E.5 Sensitivity to the Nominal Training Distribution

RAISE anchors its uncertainty set to the nominal instance set, so its robustness could in principle be an artifact of the particular nominal distribution used in the main experiments (Weibull, following Romera-Paredes et al. (2024) and Liu et al. (2024)). We test this directly by repeating the entire training procedure with Normal and Exponential nominal datasets, holding every other setting fixed, and evaluating on the same five shifted families.

Table 29: Sensitivity to the nominal training distribution on online bin packing at $\varepsilon = 0.1$. Average excess-bin ratio (%, lower is better) over all 60 OOD datasets. The Weibull RAISE value averages three runs; Normal and Exponential each use one run. Baselines use the same reported heuristics as in the main experiments; EoH-S uses diverse training instances. **Bold**: best.

Method	Avg. shifted
EoH	5.452
ReEvo	5.075
PartEvo	4.045
MoH	5.772
EoH-S	2.116
RAISE (nominal: Weibull)	2.048
RAISE (nominal: Normal)	**1.985**
RAISE (nominal: Exponential)	2.848

Table 29 shows meaningful dependence on the nominal training distribution. Normal gives the best result (1.985%), followed by Weibull (2.048%), while Exponential degrades to 2.848%. All three remain ahead of the standard fixed-nominal AHD baselines, but the Exponential run does not outperform EoH-S (2.116%). The spread (1.985–2.848%) should therefore be read as residual dependence on where the ε-ball is centred, not as nominal-distribution invariance.

E.6 Scope of the Search-Space Prior

We state the assumption RAISE makes and the assumption it removes, because the distinction is easy to blur.

What RAISE removes. It does not require the user to specify the deployment distributions, to enumerate the possible shifts, or to supply a diverse training set covering them. This is the requirement that EoH-S (Liu et al., 2026) and MoH (Shi et al., 2026) impose, and in realistic settings the challenging distributions are unknown and may only emerge after deployment. The perturbation applied at each refresh in RAISE is produced by optimization, not by specification.

What RAISE still requires. A task-specific encoding — here, scaled item sizes, operation durations, customer demands, or spatial-generator parameters — and a perturbation family over that encoding, here a signed combination of nine basis functions. These are a search-space prior. In a domain where a meaningful encoding or basis family is difficult to specify, the present instantiation would require additional design work before it could be applied. We do not claim the nine bases are universally suitable, and they are not intended to approximate the true deployment distributions; motivated by basis-function constructions (Lee et al., 2013), they define a compact space in which the LLM-free inner loop can search cheaply.

The direct route to removing it. The two components of RAISE are separable. The ε-constrained inner search does not depend on how the perturbation is parameterized, only on candidate encodings being comparable with nominal encodings. One possible extension would replace the hand-designed perturbation functions with a learned generator, such as a normalizing flow or VAE. Its output would still need projection and checking in the specified encoding space, rather than an unconstrained search in latent space. A further task decoder would retain the distinction between encoded-space feasibility and realized-instance feasibility described in Appendix B.2; the latter would require an additional preservation argument or post-decoding check.

F RAISE HEURISTIC FUNCTIONS: CODE AND ROBUSTNESS ANALYSIS

This appendix presents Python heuristic functions synthesized by RAISE across five task settings: Online Bin Packing (OBP), Online Job Shop Scheduling (JSP), Online Vehicle Routing (VRP), offline Job Shop Scheduling (JSS), and offline Traveling Salesperson Problem (TSP) with Guided Local Search (GLS). For OBP, where the reported values average three independent runs, we show the heuristic from the first $\varepsilon = 0.2$ run as a representative example. The two offline heuristics use $\varepsilon = 0.1$ and are those evaluated in Table 4. For each task we list code produced by the LLM evolutionary loop and provide a mechanistic analysis of its design choices.

F.1 ONLINE BIN PACKING

F.1.1 HEURISTIC CODE

The heuristic is a `priority(item, bins)` function: given an incoming item of size `item` and a vector `bins` of residual capacities of open bins, it returns a score vector whose *argmax* selects the bin to use (negative infinity for infeasible bins).

```python
import numpy as np
def priority(item: float, bins: np.ndarray) -> np.ndarray:
    bins = np.asarray(bins, dtype=float)
    eps = 1e-12

    if bins.size == 0:
        return bins

    residual = bins - item
    feasible = residual >= 0

    max_bin = np.nanmax(bins) + eps
    mean_bin = np.nanmean(bins) + eps

    # Smaller leftover is better.
    tight = 1.0 / (1.0 + residual)

    # Prefer a reusable leftover on the scale of the current item.
    balance = 1.0 / (1.0 + np.abs(residual - item) / (item + eps))

    # Preserve relatively large bins unless they are a very good fit.
    scarcity = 1.0 - (bins / max_bin)
    scarcity = np.clip(scarcity, 0.0, 1.0)

    # Prefer bins below the current mean remaining capacity.
    pressure = 1.0 / (
        1.0 + np.maximum(bins - mean_bin, 0.0) / (mean_bin + eps)
    )

    # Emphasize exact fits.
    exact = np.where(np.abs(residual) <= 1e-9 * max_bin, 2.0, 1.0)

    score = (
        exact
        * (0.9 * tight + 0.4 * balance)
        * (0.8 + 0.2 * scarcity)
        * pressure
    )
    return np.where(feasible, score, -np.inf)
```

Listing 1: Representative RAISE heuristic from the first $\varepsilon = 0.2$ Online Bin Packing run.

F.1.2 Robustness Analysis

Five design choices in Listing 1 help explain its behavior across Uniform, Normal, Lognormal, Exponential, and Triangular test distributions.

(1) Feasibility is enforced before ranking. The heuristic assigns $-\infty$ to every bin whose remaining capacity is smaller than the incoming item. All subsequent signals therefore compare only valid placements and cannot trade constraint violation against a favorable score.

(2) Tight packing remains the dominant signal. The reciprocal term $1/(1+r)$, where r is the residual capacity after placement, favors small leftovers and recovers the central Best-Fit intuition. The separate exact-fit multiplier doubles the score when the residual is numerically zero, making exact fills robust to the secondary terms.

(3) Leftover reuse is scaled to the observed item. The balance term rewards a post-placement residual close to the current item size. Because it uses $|r-a|/(a+\epsilon)$ rather than a fixed capacity threshold, it adapts to the local item scale and can preserve a slot likely to accept another item of comparable magnitude.

(4) Large residual-capacity bins are preserved. The scarcity factor mildly prefers smaller open bins, while the pressure term penalizes bins whose remaining capacity lies above the current mean. Together they discourage consuming a large flexible bin when a tighter feasible bin exists, without overriding an exact or near-exact fit.

(5) Multiplicative gating prevents one weak cue from dominating. Fit quality, scarcity, and packing pressure are combined multiplicatively. A placement must therefore be acceptable on all three axes to receive a high score; a large value from one term cannot fully compensate for a poor value from another. The only normalizations use the current item and current open-bin statistics, avoiding dependence on a named test distribution.

F.2 Online Job Shop Scheduling

F.2.1 Heuristic Code

The heuristic is a `priority(current_time, candidate_operations)` function: given the current simulation clock and a matrix of eligible operations (columns: machine_id, duration, release_time, due_date, remaining_work, remaining_operations), it returns a score vector whose argmax selects the operation to dispatch next.

```python
import numpy as np
def priority(current_time: int,
             candidate_operations: np.ndarray) -> np.ndarray:
    ops = np.asarray(candidate_operations, dtype=float)
    if ops.size == 0:
        return np.array([], dtype=float)

    duration           = ops[:, 1]
    release_time       = ops[:, 2]
    due_date           = ops[:, 3]
    remaining_work     = ops[:, 4]
    remaining_operations = ops[:, 5]

    start_time      = np.maximum(current_time, release_time)
    completion_time = start_time + duration
    slack           = due_date - completion_time

    # Urgency: small/negative slack gets high priority
    urgency = np.where(
        slack < 0,
        1.0 + (-slack) / (1.0 + duration),   # overdue: grows with
            tardiness
```

```
        1.0 / (1.0 + slack)                    # on-time: diminishing
            as slack grows
    )

    # SRPT-like finishing: favour small remaining work
    finishing = 1.0 / (1.0 + remaining_work)

    # Downstream risk: more remaining ops => higher priority
    ops_density = remaining_operations / (1.0 + remaining_operations)

    # Duration penalty: penalise long operations unless urgent
    duration_penalty = duration / (1.0 + remaining_work)

    w_urgency = 3.0
    w_finish  = 2.0
    w_ops     = 1.5
    w_dur     = 0.75

    score = (w_urgency * urgency
             + w_finish  * finishing
             + w_ops     * ops_density
             - w_dur     * duration_penalty)

    score = np.nan_to_num(score, nan=0.0,
                          posinf=np.finfo(float).max,
                          neginf=np.finfo(float).min)
    return score
```

Listing 2: RAISE heuristic for Online Job Shop Scheduling.

F.2.2 Robustness Analysis

(1) Dynamic, state-derived urgency. Slack = due_date − (max(current_time, release_time) + duration) is recomputed at every scheduling step from the actual system clock and operation data. No distributional assumption enters: whether job arrival times follow a Poisson process, a bursty heavy-tail, or a periodic pattern, the urgency signal correctly reflects the *current* tightness. This is in contrast to heuristics that encode timing expectations calibrated to the training distribution, which degrade when inter-arrival statistics change.

(2) Asymmetric urgency for overdue vs. on-time jobs. When slack ≥ 0, urgency $= 1/(1 +$ slack$)$ decreases gently as slack grows, de-prioritising comfortable jobs. When slack < 0, urgency $= 1 + |$slack$|/(1 +$ duration$)$ *increases* linearly with tardiness while scaling inversely with duration—so a short overdue operation gets more additional priority than a long one that is barely overdue. This asymmetry is correct regardless of the due-date distribution: it never lets an already-late job be permanently deprioritised by a deeply-backlogged queue.

(3) SRPT-like finishing component. Shortest Remaining Processing Time (SRPT) is optimal for minimising mean completion time in preemptive single-machine scheduling with known processing times and performs robustly across exponential, uniform, and heavy-tailed service-time distributions. The term $1/(1+$remaining_work$)$ approximates SRPT for the multi-machine online setting, providing a distribution-independent backbone.

(4) Downstream cascade risk. ops_density $= R/(1+R)$ where R is the number of remaining operations monotonically increases with R but saturates below 1. Prioritising jobs with more remaining operations reduces the risk of blocking downstream machines—a property whose value is independent of whether job sizes are drawn from a uniform or heavy-tailed distribution.

(5) Duration penalty modulated by remaining work. duration_penalty $= d/(1+$remaining_work$)$ penalises long operations only when there is little remaining

work left in the job. A job with large remaining work is less penalised even if its current operation is long, because blocking that machine is justified by the downstream workload. This modulation prevents the heuristic from systematically disfavouring heavy jobs, which are more prevalent under Lognormal or Pareto arrival distributions.

F.3 Online Vehicle Routing

F.3.1 Heuristic Code

The heuristic is a `priority(customer_features, candidate_insertions)` function. `customer_features` encodes the arriving customer (demand, progress, etc.); `candidate_insertions` is a matrix of feasible insertion positions with columns: route_id, position, delta_distance, residual_capacity, current_load, route_size, customer_dist, projected_route. The function returns a score vector whose argmax selects the best insertion.

```python
import numpy as np
def priority(customer_features: np.ndarray,
             candidate_insertions: np.ndarray) -> np.ndarray:
    import numpy as np

    if candidate_insertions is None or candidate_insertions.size == 0:
        return np.array([], dtype=float)

    cand = np.asarray(candidate_insertions, dtype=float)

    delta_distance    = cand[:, 2]   # incremental distance cost
    residual_capacity = cand[:, 3]   # remaining capacity after
        insertion
    current_load      = cand[:, 4]   # route utilisation in [0, 1]
    route_size        = cand[:, 5]   # normalised number of stops
    customer_dist     = cand[:, 6]   # normalised distance to
        insertion point
    projected_route   = cand[:, 7]   # normalised projected route
        length

    demand   = float(customer_features[2])
    progress = float(np.clip(customer_features[4], 0.0, 1.0))

    # Adaptive weights: shift from capacity/balance to travel cost as
        routes fill
    w_travel    = 0.3 + 1.0 * progress       # increases with progress
    w_projected = 0.1 + 0.6 * progress       # increases with progress
    w_prox      = 0.5                         # fixed proximity weight
    w_capacity  = 0.2 + 1.0 * (1.0 - progress) # decreases with
        progress
    w_balance   = 0.1 + 0.8 * (1.0 - progress) # decreases with
        progress
    w_size      = 0.4                         # fixed route-size
        penalty

    # Load-balance score: peaks at 50% utilisation
    balance_score = 1.0 - 2.0 * np.abs(current_load - 0.5)

    # Demand scaling: larger demands amplify capacity importance
    demand_scale = 1.0 + 0.5 * max(0.0, demand)

    score = (
        w_travel    * (-delta_distance)
        + w_projected * (-projected_route)
        + w_prox      * (-customer_dist)
        + w_capacity  * residual_capacity * demand_scale
        + w_balance   * balance_score
        - w_size      * route_size
    )
```

```
    return np.asarray(score, dtype=float)
```

Listing 3: RAISE heuristic for Online Vehicle Routing.

F.3.2 Robustness Analysis

(1) Progress-adaptive weighting. Four of the six weights in Listing 3 vary with `progress` $\in [0, 1]$, which measures how far route construction has advanced (fraction of customers already assigned). At `progress`=0: w_{travel}=0.3, w_{capacity}=1.2, w_{balance}=0.9. At `progress`=1: w_{travel}=1.3, w_{capacity}=0.2, w_{balance}=0.1. This *phase transition* is triggered by the actual construction state, not by any distributional parameter. It correctly reflects the decision-making reality: when routes are nearly empty, placing high-demand customers in capacity-rich routes prevents infeasibility regardless of the demand distribution; when routes are nearly full, minimising incremental travel cost is paramount.

(2) Multi-objective coverage. The score balances five complementary signals—immediate travel cost, projected route length, proximity, residual capacity, load balance—plus a route-size penalty. These signals respond to different aspects of the current routing state. For example, under a heavy-tailed demand distribution the demand-scaled capacity term $w_{\text{capacity}} \times$ `residual` $\times$ `demand_scale` strongly steers large-demand customers away from nearly-full routes, preventing constraint violations; under a spatially clustered instance the proximity and projected-route terms guide insertion toward the geographically nearest feasible route.

(3) Demand-adaptive capacity scaling. `demand_scale` $= 1 + 0.5 \times \max(0, d)$ amplifies the importance of capacity whenever the arriving customer carries a large demand. Rather than relying on the expected demand from a training distribution, this adapts to the *observed* demand value at run time. Under heavy-tailed demand distributions, where occasional very-large customers appear, this prevents catastrophic insertions that leave a route with insufficient capacity for subsequent customers.

(4) Balance score peaks at 50% utilisation. The term $1 - 2|$`current_load` $- 0.5|$ equals 1 at half-load and 0 at completely empty or fully loaded routes. The fixed 50% target favors moderate loads without assuming a particular demand distribution. Heuristics that use a distribution-specific optimal fill level degrade whenever the actual distribution shifts.

(5) Route-size penalty encourages spreading. Penalising large `route_size` discourages concentrating many stops on a single vehicle. This implicit load-balancing across routes is beneficial across all distributions because it maintains spare capacity on multiple routes simultaneously, reducing the chance of total infeasibility when a burst of high-demand customers arrives—a concern particularly acute under heavy-tailed or bursty demand distributions.

F.4 Offline Job Shop Scheduling

F.4.1 Heuristic Code

Unlike the online dispatching rule, `solve(jobs, n_machines)` receives all jobs, including their ordered operations, release times, and due dates, and constructs a complete schedule. The returned nested list stores the start time of each operation, indexed by job and operation. Listing 4 preserves the evaluated function, including unused bookkeeping and the unused `lb_completion` helper; these do not affect scheduling decisions.

```
import numpy as np
def solve(jobs: list, n_machines: int) -> list:
    n_jobs = len(jobs)
    if n_jobs == 0:
        return []

    job_ops = []
    release = [0] * n_jobs
    due = [0] * n_jobs
```

```
    job_total = [0] * n_jobs
    tail = []

    for j, job in enumerate(jobs):
        ops = job.get("operations", [])
        job_ops.append(ops)
        release[j] = int(job["release_time"])
        due[j] = int(job["due_date"])

        t = [0] * (len(ops) + 1)
        total = 0
        for k in range(len(ops) - 1, -1, -1):
            total += int(ops[k]["duration"])
            t[k] = total
        tail.append(t)
        job_total[j] = total

    schedule = [[0] * len(ops) for ops in job_ops]

    machine_free = [0] * n_machines
    job_ready = release[:]
    next_op = [0] * n_jobs
    remaining = sum(len(ops) for ops in job_ops)

    # Prefer early starts and long remaining chains.
    machine_load = [0] * n_machines
    for j in range(n_jobs):
        for op in job_ops[j]:
            machine_load[int(op["machine_id"])] += int(op["duration"])

    def lb_completion(j, k, start_time):
        return (start_time + int(job_ops[j][k]["duration"])
                + tail[j][k + 1])

    while remaining > 0:
        best = None
        best_j = -1
        best_k = -1
        best_start = 0

        for j in range(n_jobs):
            k = next_op[j]
            if k >= len(job_ops[j]):
                continue

            op = job_ops[j][k]
            m = int(op["machine_id"])
            d = int(op["duration"])

            start = job_ready[j]
            if machine_free[m] > start:
                start = machine_free[m]

            chain = tail[j][k]
            bottleneck = max(machine_free[m], job_ready[j])
            density = chain / float(max(1, d))
            urgency = due[j] - (job_ready[j] + chain)
            future_pressure = machine_load[m]

            key = (
                bottleneck,
                -density,
                -chain,
                urgency,
                future_pressure,
```

```
                d,
                j,
                k,
            )

            if best is None or key < best:
                best = key
                best_j = j
                best_k = k
                best_start = start

        op = job_ops[best_j][best_k]
        m = int(op["machine_id"])
        d = int(op["duration"])

        actual_start = best_start
        if actual_start < job_ready[best_j]:
            actual_start = job_ready[best_j]
        if actual_start < machine_free[m]:
            actual_start = machine_free[m]

        end = actual_start + d
        schedule[best_j][best_k] = int(actual_start)
        job_ready[best_j] = end
        machine_free[m] = end
        next_op[best_j] += 1
        remaining -= 1

    return schedule
```

Listing 4: RAISE heuristic for offline Job Shop Scheduling ($\varepsilon = 0.1$).

F.4.2 Robustness Analysis

(1) Feasibility through construction. Only the next unscheduled operation of each job is considered. Its start time is the maximum of the job's ready time and the machine's next available time. Updating both after each assignment enforces release times, job precedence, and machine non-overlap for valid inputs with integer processing times. These constraints are enforced independently of the processing-time distribution, rather than through a learned penalty.

(2) Lexicographic, state-dependent priorities. The tuple `key` is minimized lexicographically, not as a weighted sum. The earliest feasible start is therefore the dominant criterion. Ties favor a larger remaining-chain-to-operation-duration ratio, then a longer remaining chain, smaller slack, lower machine load, and shorter duration; job and operation indices break any remaining ties deterministically. The chain-based criteria can prioritize operations that unlock substantial downstream work, but they cannot override an earlier feasible start.

(3) Relative workload signals. The dimensionless `density` uses the observed remaining processing time relative to the current operation's duration, without fitting a distribution-specific threshold. The slack term likewise uses the actual due date and remaining chain. Importantly, `machine_load` is the total processing load computed once from the input, not a dynamically updated residual load. It affects decisions only after the preceding tuple entries tie.

(4) Evidence and limitations. These structural choices offer a plausible explanation for transfer across processing-time families, but do not guarantee optimal makespan: the heuristic commits greedily and does not backtrack or insert operations into earlier machine idle gaps. Table 4 reports an average makespan ratio of 1.2129, compared with 1.2536 for the same synthesis pipeline without the inner loop, with improvements on all five families. This comparison evaluates the complete heuristic; it does not isolate the causal contribution of individual priority terms.

F.5 Offline Traveling Salesperson Problem with Guided Local Search

F.5.1 Heuristic Code

The evolved function `update_edge_distance` modifies the distance matrix used to guide a classical GLS solver; it does not directly construct a tour. Its inputs are the original distance matrix, the current locally optimized tour, and a symmetric matrix of historical edge-penalization counts maintained by the solver. The returned matrix guides perturbation moves, while incumbent tours are evaluated using the original distances. Listing 5 preserves the evaluated function's logic, with comments and line wrapping adjusted for readability. The local variable `eps` is a numerical safeguard, not the adversarial-search radius $\varepsilon = 0.1$.

```python
import numpy as np

def update_edge_distance(edge_distance: np.ndarray,
                         local_opt_tour: np.ndarray,
                         edge_n_used: np.ndarray) -> np.ndarray:
    import numpy as np
    # Defensive copies and basic checks.
    D = np.array(edge_distance, dtype=float, copy=True)
    n = D.shape[0]
    assert D.shape == (n, n)
    tour = np.asarray(local_opt_tour, dtype=int)
    assert tour.ndim == 1 and tour.size == n
    counts = np.array(edge_n_used, dtype=float, copy=False)
    assert counts.shape == (n, n)

    eps = 1e-12

    # Build the undirected tour-edge mask.
    a_nodes = tour
    b_nodes = np.roll(tour, -1)
    tour_mask = np.zeros((n, n), dtype=bool)
    tour_mask[a_nodes, b_nodes] = True
    tour_mask[b_nodes, a_nodes] = True

    # Utility combines length, local context, and penalty history.
    gamma = 2.0
    edge_infos = []
    for i in range(n):
        u = tour[i]
        v = tour[(i + 1) % n]
        prev = tour[(i - 1) % n]
        nxt = tour[(i + 2) % n]

        l_uv = D[u, v]
        l_prev = D[prev, u]
        l_next = D[v, nxt]

        context = (l_prev + l_uv + l_next) / 3.0
        # Bonus when an edge is shorter than its local context.
        cluster_bonus = max(0.0, context / (l_uv + eps) - 1.0)

        util = (l_uv * (1.0 + gamma * cluster_bonus)
                / (1.0 + counts[u, v]))
        edge_infos.append((u, v, l_uv, util))

    utils = np.array([e[3] for e in edge_infos])
    mean_util = np.mean(utils) if utils.size > 0 else 0.0

    # Augment the top 20% of tour edges (at least one).
    k = max(1, int(0.20 * n))
    top_idx = utils.argsort()[::-1][:k]

```

```
    # Scale the penalty magnitude to the current instance.
    mean_edge = (np.mean(D[np.triu_indices(n, 1)])
                 if n > 1 else float(D[0, 0]))
    base_lambda = max(eps, 0.20 * mean_edge)

    penalty = np.zeros((n, n), dtype=float)
    for idx in top_idx:
        u, v, l_uv, util = edge_infos[idx]
        inc = base_lambda * (1.0 + util / (mean_util + eps))
        penalty[u, v] += inc
        penalty[v, u] += inc

    # Reward non-tour edges with low penalization counts.
    reward = np.zeros((n, n), dtype=float)
    median_used = (np.median(counts[np.triu_indices(n, 1)])
                   if n > 1 else 0.0)
    beta = 0.05
    iu, ju = np.triu_indices(n, 1)
    for i, j in zip(iu, ju):
        if not tour_mask[i, j] and counts[i, j] <= median_used:
            rew = -beta * mean_edge / (1.0 + counts[i, j])
            reward[i, j] += rew
            reward[j, i] += rew

    updated = D + penalty + reward
    updated = np.maximum(updated, eps)
    updated = 0.5 * (updated + updated.T)

    return updated
```

Listing 5: RAISE guide heuristic for offline TSP with GLS ($\varepsilon = 0.1$).

F.5.2 Robustness Analysis

(1) Local geometry and search history. For each tour edge, the utility combines its length with a bonus determined by the mean length of that edge and its two neighbors along the tour. An edge shorter than this local context receives a positive bonus, allowing the guide to target locally short edges as well as globally long ones. The name `cluster_bonus` denotes this local length contrast, not an explicit clustering algorithm. Dividing by one plus the historical penalization count reduces repeated emphasis on previously targeted edges.

(2) Instance-relative perturbations. The guide adds penalties to the $\max(1, \lfloor 0.2n \rfloor)$ highest-utility tour edges. Penalty magnitudes use 0.2 times the mean off-diagonal distance and each edge's utility relative to the mean tour-edge utility. This makes the guide responsive to instance size and distance scale rather than fixed absolute distance thresholds. Ignoring the numerical safeguard, uniformly scaling all distances scales the returned matrix by the same factor. The outer GLS solver then selects the five most augmented edges for focused perturbations and increments their counts; the top-20% guide set is not itself the set of counter updates.

(3) Exploration with symmetric, positive guide weights. Non-tour edges whose counts do not exceed the median receive a small distance reduction, scaled by 0.05 times the mean distance and attenuated by their counts. This encourages consideration of alternatives with low penalization history, though a low count does not necessarily mean an edge has never been visited. The final clipping and symmetrization keep guide weights positive and symmetric. Tour feasibility is maintained by the solver's 2-opt and relocate moves, not by these matrix operations; the modified weights need not satisfy the triangle inequality.

(4) Evidence and limitations. Table 4 reports mean gaps on 47 TSPLIB instances of 0.4932% for RAISE, 0.5447% without the inner loop, and 3.3565% for classical GLS. Local geometric ratios and history-dependent exploration are plausible transfer mechanisms, but have not been individually

ablated. The guide changes search decisions, not the tour objective, and provides no optimality or out-of-distribution performance guarantee.

G COMPUTATIONAL COST ANALYSIS

This appendix quantifies the LLM API request budget, token consumption, and evaluation overhead of RAISE, and benchmarks these costs against five competing LLM-based AHD frameworks. All LLM-call counts follow the unified *sample budget* convention used across the comparison (each budget unit = one call to the LLM API resulting in one candidate heuristic program).

G.1 RAISE COST BREAKDOWN

Table 30 lists the cost of a single RAISE run at the standard budget of $N_{\max} = 1000$ LLM samples, using a GPT-5-mini class model used in our experiments.

Table 30: RAISE per-run cost breakdown ($N_{\max}$=1000, OBP primary configuration). Token counts are computed from the system design (Appendix B); API costs use published list prices (GPT-5-mini: \$0.25/\$2.00 per 1M in/out tokens). Inner adversarial search uses CPU only (no LLM calls).

Component	**Description**	**Quantity**
LLM API Calls (Outer Evolutionary Loop)		
Total budget $N_{\max}$	Hard limit on LLM calls	1,000
— CREATE	New heuristic	≈20
— E1	Crossover	≈245
— E2	Backbone extraction + Crossover	≈245
— M1	Mutation	≈245
— M2	Mutation	≈245
Token Consumption per LLM Call (approximate averages)		
CREATE	Task desc. + template	1,200 in / 500 out
E1	Task + 2 parents + instruction + template	2,850 in / 650 out
E2	Task + 2 parents + instruction + template	3,050 in / 650 out
M1	Task + parent + instruction + template	1,850 in / 600 out
M2	Task + parent + instruction + template	1,650 in / 600 out
Weighted average		**2,200 in / 620 out**
Total Token Budget (1,000 calls)		
Input tokens	$1,000 \times 2,200$	≈2.2M
Output tokens	$1,000 \times 620$	≈0.62M
Estimated API Cost		
GPT-5-mini	\$0.25/\$2.00 per 1M in/out	≈\$ 1.79
Inner Adversarial Search Overhead (CPU only; no LLM)		
instance refreshes	Every τ=5 gens; 200 gens total	40
Inner evaluations per refresh	P_{in}=8 pop × G_{in}=4 gens	32
Re-scoring after refresh	$P_{max} = 10$ programs	40
Outer evaluations per generation	$P_{max} = 10$ programs × 15 instances	150
Total evaluations	$40\times(32+40) + 10\times15\times100 =$	≈ 17,880
Wall-Clock Time (OBP, 20 parallel samplers/evaluators)		
Measured runtime		≈2.0 h

G.2 COMPARISON WITH EXISTING FRAMEWORKS

Table 31 compares RAISE against EoH, ReEvo, PartEvo, MoH, and EoH-S at $N_{\max}$=1000 on the online bin packing problem.

Token overhead. All methods share the same 1,000-call budget. RAISE's total token cost (≈2.82M) is comparable to EoH (≈2.80M). ReEvo and PartEvo are substantially more expensive in tokens (≈4.30M and ≈4.51M respectively) due to their multi-stage reflection and critique prompting strategies, which usually generate longer and more elaborate heuristic programs.

Table 31: Per-run LLM cost comparison across AHD frameworks at $N_{\max}$=1000 samples. † Estimated from published architecture. API cost uses GPT-5-mini: \$0.25/\$2.00 per 1M in/out tokens

Method	LLM Calls	Total Tokens	Est. Cost (mini)	#Instance Evals
EoH (Liu et al., 2024)	1,000	≈2.80M	≈\$ 1.78	≈5,000
ReEvo (Ye et al., 2024)	1,000	≈4.30M	≈\$ 2.74	≈5,000
PartEvo (Hu & Zhang, 2025)	1,000	≈4.51M	≈\$ 2.89	≈5,000
MoH (Shi et al., 2026)†	1,000	≈3.10M	≈\$ 1.95	≈10,000
EoH-S (Liu et al., 2026)	1,000	≈2.86M	≈\$ 1.81	≈128,000
RAISE (ours)	1,000	≈2.82M	≈\$ 1.79	≈17,880

EoH-S incurs a hidden evaluation overhead. EoH-S has a token cost comparable to RAISE, but its instance evaluation count (≈128,000) is about 7.2 times the stated RAISE estimate and exceeds the EoH, ReEvo, and PartEvo counts by over an order of magnitude. This arises from its *post-hoc portfolio selection* step, in which the full heuristic population is evaluated across a diverse set of 128 training instances spanning multiple distributions to select the top-K performers. Beyond the evaluation cost, this step requires **prior knowledge of deployment distributions**—often unavailable in practice. RAISE requires only nominal training-distribution instances, and delivers a single deployable heuristic rather than a portfolio.

MoH incurs dual-task evaluation overhead. MoH's ≈10,000 instance evaluations (double that of EoH) stem from its simultaneous training on two task variants—e.g., OBP under two item distributions—requiring every candidate heuristic to be scored on both variants each generation. This broadens coverage across the two training distributions but provides no adversarial robustness mechanism.

Inner adversarial search is essentially free at the API level. RAISE's inner worst instance search accounts for additional heuristic evaluations in CPU time only, which is independent of the number of deployment distributions, in contrast to EoH-S's evaluation overhead, which scales linearly with the size of its 128-instance training set.