UCE: UNCERTAINTY CALIBRATED EVOLUTION FOR AUTOMATIC HEURISTIC DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

In iterative LLM-based heuristic design, evaluation scores serve not only to rank candidate implementations, but also to shape the search directions that an LLM continues to explore or abandons. However, a raw score evaluates only one implementation in isolation and can be misleading when interpreted without considering the performance of other candidates generated from the same parent. In particular, an isolated low score may cause the LLM to prematurely discard a potentially productive search direction. We introduce Uncertainty Calibrated Evolution (UCE), a stage-aware scoring paradigm that calibrates candidate evaluations according to the collective performance within each fixed-parent stage. Specifically, UCE contextualizes each candidate's score using the mean and dispersion of its stage-level candidate pool, transforming an isolated evaluation into a relative signal for subsequent selection and feedback. This design reduces the disproportionate influence of low or otherwise atypical scores on the LLM's iterative decisions, without modifying the program generator or requiring additional model training. We evaluate UCE across diverse operations-research datasets and LLM-AHD baselines. UCE improves overall performance and reduces premature acceptance and rejection in counterfactual replay, suggesting that stage-level score calibration can provide more reliable guidance for iterative heuristic search.

1 INTRODUCTION

LLM-based automatic heuristic design (LLM-AHD) uses a large language model to write and iteratively improve an algorithm for a downstream optimization problem, such as vehicle routing, scheduling, and packing in operations research, automated planning, or mathematical problem solving (Liu et al., 2024; Romera-Paredes et al., 2024). The objective is to obtain a heuristic that can outperform a human-designed baseline with little or no human intervention. Given a problem specification and an evaluator, the model proposes executable code, runs it on training instances, and uses the program, its score, execution feedback, and selected earlier attempts as context for the next proposal. Each round therefore couples program generation, evaluation, and selective reuse of experience. Designing this loop so that a model can use a finite evaluation budget to iteratively produce high-quality algorithms is a central research question (Romera-Paredes et al., 2024; Liu et al., 2024).

One line of work focuses on balancing exploration and exploitation. Exploration allocates part of the budget to diverse or uncertain candidates, while exploitation concentrates the budget on candidates that currently score well. FunSearch and EoH evolve programs in this way, and MCTS-AHD organizes the trade-off with a search tree (Romera-Paredes et al., 2024; Liu et al., 2024; Zheng et al., 2025). A second line of work treats each iteration as a reasoning process: reflection revises an evaluated program, while accumulated experience, hindsight, and foresight provide information for later revisions. ReEvo and HiFo-Prompt are examples of this direction (Ye et al., 2024; Chen et al., 2026). Despite these different mechanisms, both lines depend on scores to make the search operate. Figure 2 shows why this dependence is potentially fragile. Panels (a) and (b) contain substantial score noise, including many low scores from early exploration and from stochastic LLM implementation. Panel (c) shows that deeper search more often produces a program above the target cutoff, which is consistent with the usual intuition that continued reasoning can improve an idea. The overlooked question is whether an early low score can influence later generation strongly enough to prevent that improvement.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

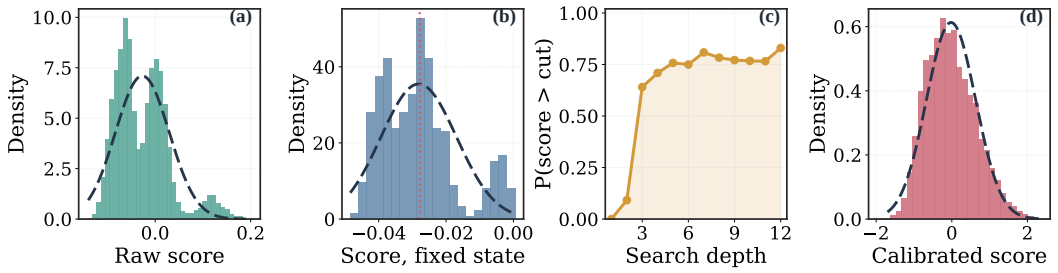

Figure 2: Variation in program scores during search. From left to right: (a) pooled raw scores have multiple modes; (b) revisions from a fixed parent vary in score; (c) deeper search more often yields a score above a target cutoff; and (d) calibrated scores are shown with a descriptive Gaussian reference. Panel (c) is a conceptual breadth/depth example and does not depict UCE's search policy.

These observations expose two challenges. **Challenge 1: parent selection is score-dependent.** Most methods use a parent program to keep the prompt manageable and to define the next local search (Liu et al., 2024; Ye et al., 2024; Zheng et al., 2025; Chen et al., 2026). The parent is chosen almost entirely by its score. During early exploration, committing a poor program can therefore move subsequent iterations into a suboptimal region even when later revisions from that program could have improved. **Challenge 2: a score describes the implementation more than the design.** It is noisy, reflects the current program on the current evaluation set, and largely ignores the quality of the underlying thought as well as the model's ability to realize that thought in code. We therefore need a unified score that considers the model's stage, observed performance, and information about the design process.

We propose Uncertainty Calibration Evolution (UCE) to provide this context. We assume stage-conditional independence: once the current parent state and its prompt context are fixed, earlier stages affect the current stage only through that state. UCE consequently focuses on the local transition from one parent to the next parent, rather than trying to model the full search history. For a local problem, we expect the model's behavior to become relatively stable and to approach a local regime within a truncated search interval, consistent with the local view used in self-adapting evolutionary search (Hansen & Ostermeier, 2001). We use the mean and dispersion observed in that interval to define a compact Gaussian reference. Intuitively, this reduces the penalty assigned to zero or otherwise poor outcomes, so a noisy early result is less likely to dominate the search; calibrated scores then vary less in response to isolated failures and stay closer to the stage's typical behavior. Most search mechanisms already accept strong scores directly, so UCE does not redesign the treatment of high-scoring programs. Instead, such programs are checked on a holdout set independent of both train selection and test reporting, since an isolated high training score should not be trusted without an independent evaluation.

Our contributions are as follows.

- We identify a limitation of score-based selection in LLM-AHD: a score measures current performance but also guides decisions about whether to continue refining a program.
- We introduce UCE, a training-free, stage-wise score calibration that uses the mean and dispersion of scores from a common parent to inform candidate selection and feedback for later revisions.
- We evaluate UCE against two baselines on twelve operations-research tasks and five LLM-AHD baselines in nine from-scratch settings. UCE improves overall performance and reduces both premature acceptance and premature rejection during search.

2 Related Work

Self-evolving LLM agents. Self-evolving agents use feedback from earlier attempts to improve later outputs or behavior. Self-Refine revises answers using LLM-generated feedback, Reflexion retains verbal reflections, and CRITIC uses external tools to identify and correct errors (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024). Other methods update the information or programs that guide an agent: GEPA optimizes instructions, ACE maintains reusable contextual knowledge,

and RecHarness guides revisions to recommender agents (Agrawal et al., 2025; Zhang et al., 2025c; Ling et al., 2026). ADAS and the Darwin Gödel Machine extend this process to the design of executable agents (Hu et al., 2024; Zhang et al., 2025a). These studies motivate reusing experience across iterations. UCE examines a specific part of this process: how performance scores should inform the selection of programs and the feedback used to revise them.

LLM-based automatic heuristic design. FunSearch, EoH, and EoH-S combine program generation with evolutionary selection to discover heuristics for optimization problems (Romera-Paredes et al., 2024; Liu et al., 2024; 2026). ReEvo adds reflections on candidate performance, while HiFo-Prompt uses hindsight to retain design knowledge and foresight to guide exploration (Ye et al., 2024; Chen et al., 2026). MCTS-AHD retains a tree of candidate programs, allowing the search to revisit alternatives that are not currently the best (Zheng et al., 2025). Further work studies joint adaptation of algorithms and query strategies, structural priors, and novelty-based or bandit-based selection (Huang et al., 2026; Li et al., 2025; Lange et al., 2025). Multi-objective search, Pareto-grid search, and long-horizon benchmarks extend AHD to multiple objectives and longer search budgets (Yao et al., 2025; Ha et al., 2026; Imajuku et al., 2025). These methods address how to explore programs and retain useful experience. UCE addresses the complementary question of how to interpret an evaluation within that search. It adjusts a candidate's score relative to the outcomes observed from the same parent before using it for selection and feedback.

Uncertainty in self-evolution. Uncertainty estimates help distinguish observed performance from the strength of the evidence supporting it. Work on predictive uncertainty studies data variability and uncertainty in model estimates, including deep ensembles (Kendall & Gal, 2017; Lakshminarayanan et al., 2017). Conformal prediction provides coverage guarantees under stated assumptions, with extensions for distribution shift (Angelopoulos & Bates, 2023; Tibshirani et al., 2019; Gibbs & Candes, 2021). For LLMs, uncertainty has been estimated from token probabilities, entropy, and self-supervised signals (Ma et al., 2025; Zhang et al., 2025b), and used to weight preference feedback or allocate computation across search trajectories (Wang et al., 2024; Xing et al., 2026). Bandit-based selection and direction routing similarly guide where an evolving system spends its budget (Lange et al., 2025; Ling et al., 2026; Auer et al., 2002; Thompson, 1933). UCE focuses on variation among the scores of generated implementations. Its local mean and dispersion provide a descriptive refer-

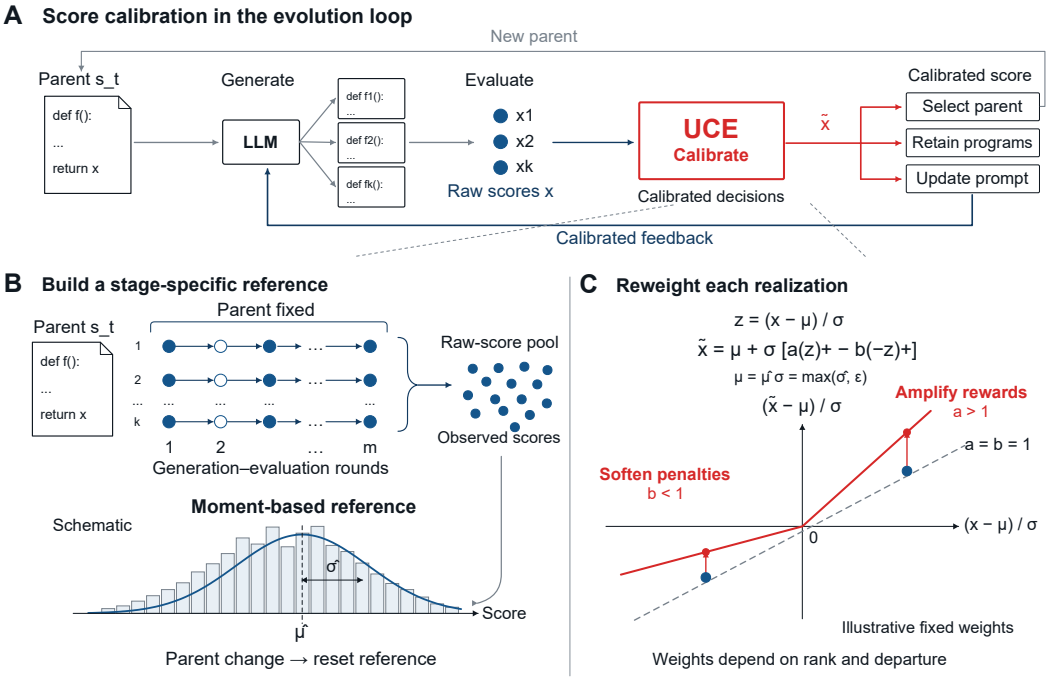

Figure 1: Overview of UCE. Programs generated from one parent form a stage. UCE calibrates their scores against the observed stage distribution before using them for selection and feedback.

ence for score adjustment, without assigning uncertainty to specific underlying causes or providing a predictive confidence interval for future performance.

3 METHOD

UCE maintains an online score reference while a parent remains fixed (Figure 1). Every raw evaluation from every generated population is added to the stage pool and to the history supplied to the LLM. After a warm-up period, UCE uses the pool mean and dispersion to adjust the decision score of low-performing candidates. High-performing candidates keep their raw score for selection and are additionally evaluated on an independent holdout set to check whether their training and holdout performance agree. When the parent changes, UCE starts a new stage with a new score pool while retaining the search history.

The formulation below separates the generation process, the statistical reference, and the score adjustment. We first specify which evaluations belong to the same stage. We then summarize all observed scores online and use that summary to calibrate low-score candidates. The reference describes performance under the current generation policy and evaluation budget. It provides context for interpreting a result, rather than directly predicting the best program that further search might discover.

3.1 DEFINING A SEARCH STAGE

Stage t is defined by a parent state s_t containing the parent program, task description, and the context passed to the generation process. Let τ_t denote the sequence of generation and evaluation steps within the stage. We treat s_t as the complete state used in the stage, so earlier stages affect current proposals only through this state,

$$p(\tau_t \mid s_t, \tau_{<t}) = p(\tau_t \mid s_t). \tag{1}$$

This assumption specifies how information is passed between stages; it does not make evaluations within a stage independent. Changing the parent creates a new state and resets the observed score pool.

Within a stage, each round generates k candidates from the fixed parent. Across rounds, index j identifies the proposal stream used by the search procedure. Let $\mathcal{H}_{t,\ell-1}$ contain the observations and feedback available before round ℓ. Candidate generation and evaluation follow

$$c_{t,j,\ell} \sim p_\theta(c \mid s_t, \mathcal{H}_{t,\ell-1}, j), \qquad x_{t,j,\ell} = S(c_{t,j,\ell}), \quad j = 1, \ldots, k, \tag{2}$$

where $c_{t,j,\ell}$ is an executable candidate and S is its evaluator score, oriented so that larger values are better. For an objective J to be minimized, we use $S = -J$. Feedback from previous rounds can change later proposals even though the parent remains fixed. The score pool therefore summarizes the outcomes of this evolving sampling process.

UCE preserves the search procedure's original ranking of candidates and uses that ranking only to determine the strength of the score adjustment. The index j identifies a proposal stream in the implementation; it does not imply that the stream represents a distinct semantic idea.

3.2 ESTIMATING THE LOCAL SCORE DISTRIBUTION

We pool the raw scores collected from the current parent to estimate the typical performance and variation among generated programs. The empirical moments after m complete rounds are

$$\begin{aligned} \widehat{\mu}_{t,m} &= \frac{1}{km} \sum_{\ell=1}^{m} \sum_{j=1}^{k} x_{t,j,\ell}, \\ \widehat{\sigma}^2_{t,m} &= \frac{1}{km} \sum_{\ell=1}^{m} \sum_{j=1}^{k} (x_{t,j,\ell} - \widehat{\mu}_{t,m})^2. \end{aligned} \tag{3}$$

The mean provides a reference level for the current stage. The standard deviation describes how widely candidate scores vary around that level, allowing the magnitude of a deviation to be interpreted in context. Both statistics describe the same observed pool. They combine variation due to design ideas, their implementations, and the evaluation procedure itself.

This reference is local to the generation policy and evaluation budget used for the current parent. It summarizes the programs actually sampled, not all programs that could in principle be written for the task. In particular, failed executions remain in the pool when the evaluator assigns them a score, because they are outcomes the search encountered. The resulting spread describes variation among implementations; it is not a confidence interval for the unknown best program. As the pool grows, its moments can be updated without storing the full history. We reset them when the parent changes so that scores from different starting programs are not treated as draws from the same search context.

For a compact numerical reference, we use the observed mean and variance to define a Gaussian reference and an interval around its center,

$$\widehat{P}_{t,m} = \mathcal{N}(\widehat{\mu}_{t,m}, \widehat{\sigma}^2_{t,m}), \qquad \mathcal{I}_{t,m}(\beta) = [\widehat{\mu}_{t,m} - \beta\widehat{\sigma}_{t,m}, \widehat{\mu}_{t,m} + \beta\widehat{\sigma}_{t,m}], \tag{4}$$

where $\beta > 0$ sets the interval width. This is a descriptive normalization device, not a generative model, an attainable-score bound, or a claim that the empirical distribution is Gaussian. The mean and dispersion define a local scale for low, typical, and unusually high results even when the empirical distribution has several modes. Appendix A describes online estimation and sampling uncertainty in the estimated mean.

3.3 Calibrating Scores within the Evolution Loop

Calibration adjusts how strongly a candidate's deviation from the stage mean affects its score. After round m, let $\mu = \widehat{\mu}_{t,m}$ and $\sigma = \max(\widehat{\sigma}_{t,m}, \varepsilon)$, where $\varepsilon > 0$ prevents division by zero. We first standardize the candidate's raw score,

$$z_{t,j,m} = \frac{x_{t,j,m} - \mu}{\sigma}. \tag{5}$$

The estimates use every raw observation available through the current round. Calibration begins only after a warm-up period that collects an initial score pool. We retain every score in the history supplied to the LLM, but apply the score correction only to candidates below the stage mean:

$$\widetilde{x}_{t,j,m} = x_{t,j,m} + (1 - b_{t,j,m})(\mu - x_{t,j,m})_+, \qquad (u)_+ = \max(u, 0). \tag{6}$$

Here $b_{t,j,m}$ is a nonnegative penalty weight that depends on the candidate's existing rank, standardized deviation, and accumulated history. For a below-mean score, $b_{t,j,m} < 1$ reduces the downward departure from the stage mean; a score at or above the mean is left unchanged. The dispersion sets the scale used to choose the penalty weight, as detailed in Appendix B. High-scoring candidates are sent to the independent holdout evaluator for an audit of train–holdout agreement rather than receiving a further score correction.

At each round, UCE adds all new raw scores from the generated population to the stage pool, updates Equation 3, and recalculates the decision scores online. All raw and calibrated outcomes are retained in the history used by subsequent prompts, while calibrated scores are never added to the raw pool as new observations. Before warm-up is complete, the search uses raw scores. When the parent changes, UCE resets the reference pool but keeps the history. The surrounding search procedure determines how these scores affect candidate retention, parent replacement, or the allocation of further evaluations.

Effect on selection. The raw score answers how well a particular program performed. The calibrated score additionally accounts for how that result compares with other programs generated from the same parent. For example, a low score within a highly variable stage need not receive the same penalty as a similarly low score within an otherwise consistent stage. This distinction can affect which programs remain available for revision and which outcomes shape later prompts. It does not establish that any particular revision will succeed. We assess its practical value through final program performance, selection diagnostics, and ablations in the next section.

Algorithm 1 shows where calibration enters a standard search loop. The generator, native ranking, and parent-update rule remain those of the underlying method. Warm-up delays calibration but does not discard any observations: every raw score enters the online stage reference and every raw or calibrated score is placed in the history. After warm-up, only below-mean scores are adjusted; high scores retain their raw values and trigger the independent holdout audit. New results from every population update the pool before the next calibration, and a parent change starts a new reference pool.

Algorithm 1 UCE in a program-evolution search

Require: Initial parent s, evaluator S, generator G, warm-up $k_{\min}$
Initialize history $\mathcal{H} \leftarrow \{\}$, raw-score pool $\mathcal{P} \leftarrow \{\}$, round $m \leftarrow 0$
while the LLM-call allowance remains **do**
  $m \leftarrow m + 1$; generate $c_{1:k} \leftarrow G(s, \mathcal{H})$
  for $j = 1, \ldots, k$ **do**
    Evaluate $x_j \leftarrow S(c_j)$; append (c_j, x_j) to $\mathcal{H}$ and x_j to $\mathcal{P}$
  end for
  Update (μ, σ) from the raw scores in $\mathcal{P}$
  Obtain ranks $r_{1:k}$ from the underlying search procedure
  for $j = 1, \ldots, k$ **do**
    if $m < k_{\min}$ **then**
      $\widetilde{x}_j \leftarrow x_j$
    else if $x_j \geq \mu$ **then**
      $\widetilde{x}_j \leftarrow x_j$; audit c_j on the independent holdout set
    else
      $z_j \leftarrow (x_j - \mu)/\max(\sigma, \varepsilon)$
      $b_j \leftarrow \mathcal{W}(r_j, z_j, \mathcal{H})$
      Compute $\widetilde{x}_j$ by Equation 6
    end if
  end for
  Append all $(c_j, \widetilde{x}_j)$ to $\mathcal{H}$; use $\widetilde{x}_{1:k}$ for selection
  Update s and $\mathcal{H}$ using the underlying search procedure
  if the parent s changed **then**
    Reset $\mathcal{P} \leftarrow \{\}$ and $m \leftarrow 0$
  end if
end while
return the program with the best raw training objective

4 EXPERIMENTS

4.1 SETUP

We use two complementary experimental settings:

- **Breadth across twelve OR tasks.** We refine the same OR-Tools incumbent on tasks spanning assignment, graph coloring, routing, scheduling, rostering, and covering. This suite tests whether score calibration helps iterative design across problem families. We compare UCE with the generation-based EoH (Liu et al., 2024) and the reflection-based ReEvo (Ye et al., 2024). Each task uses 64 training instances for program selection and an independent pool of 128 instances for final evaluation.
- **Transfer on three design problems.** We design heuristics from scratch for TSP, CVRP, and online bin packing (BPP). TSP and CVRP evolve on 20-node or 20-customer instances and test the selected program on 1000 unseen instances at each of three larger sizes ($n = 50, 100, 200$). BPP evolves on `binpack1` and tests on separate `binpack2/3/4` sets. These comparisons include FunSearch (Romera-Paredes et al., 2024), EoH, ReEvo, MCTS-AHD (Zheng et al., 2025), and HiFo-Prompt (Chen et al., 2026); backbone, evaluator, and data splits are held fixed within each comparison.

For the twelve-task suite, we report percentage improvement over the initial OR-Tools solution on each test pool. For TSP and CVRP, the objective is mean tour or route length; for BPP, it is the mean excess ratio over the capacity lower bound. TSP and CVRP use 128 training instances with separate calibration, audit, and frozen test sets; the larger-size transfer tests use newly sampled instances. BPP uses distinct fitting, calibration, and audit instances during search, then evaluates on held-out OR-Library groups. Consequently, candidate selection and final reporting use disjoint data. For each method, the common search allowance is 500 LLM calls, counting generation and reflection. Debugging may use additional calls, but a run never exceeds 600. We run three independent searches on the training split, select the best program by its training objective across the three runs, and evaluate that program on the held-out test split. This train-selection and held-out-test design is in

line with the experimental practice of EoH and ReEvo. Appendix C details the datasets and solvers; Appendix D lists method-specific search controls. We also study selection errors with counterfactual replay: after an original search rejects or commits a branch, replay continues the branch to a fixed horizon or grants prespecified extra attempts to observe whether that decision would have held. These continuations are used for diagnostics rather than final-program selection. Finally, we ablate UCE's components and test the sensitivity of its search parameters.

Table 1: Performance on the twelve-task OR suite. Each cell reports the percentage improvement over the initial OR-Tools incumbent on 128 held-out instances (higher is better). Programs are selected on the train-64 split. Bold indicates the best result for each task.

Method	QAP	Color	VRPTW	PDPTW	MDVRP	JSSP	FJSP	PFSP	RCPSP	GAP	Nurse	SCP
ReEvo	+3.14	+0.00	+11.33	+10.01	+6.73	+25.27	+0.82	+7.45	+0.58	**+1.07**	+0.12	+1.59
EoH	+3.18	−0.72	+9.05	+11.08	+7.89	+25.25	+20.89	+7.44	+2.39	+0.91	−0.08	+1.45
UCE	**+4.04**	**+7.41**	**+11.60**	**+12.65**	**+10.62**	**+26.92**	**+21.16**	**+8.49**	**+2.50**	+1.04	**+0.20**	**+1.69**

4.2 Performance across twelve OR tasks

UCE achieves the largest improvement over the OR-Tools incumbent on 11 of 12 tasks spanning assignment, coloring, routing, scheduling, rostering, and covering, and it is the only method that improves on every task (Table 1). All three methods obtain similar gains on JSSP, PFSP, and VRPTW, whereas Coloring shows a clearer difference: UCE improves on the incumbent, ReEvo matches it, and EoH falls below it. On FJSP, both UCE and EoH achieve substantial improvements, while ReEvo improves only slightly. The strongest baseline differs by task. GAP is the exception to UCE's lead, with ReEvo obtaining a slightly better result.

The evolution curves show how these outcomes develop (Figure 3). On VRPTW, PDPTW, and FJSP, EoH improves earlier, while UCE makes further gains later in the run. On Coloring and Nurse, UCE improves while the baseline curves remain close to or below the initial objective. RCPSP shows a different pattern: UCE improves early, and EoH approaches it later. Thus, UCE's advantage is not associated with a uniform pattern of early or late improvement. The curves describe the progress of the search as a whole; we examine individual selection decisions separately below.

Table 2: From-scratch comparison with the closed-source backbone. TSP and CVRP report mean tour or route length over 1000 instances per scale, and BPP reports mean excess ratio. Lower is better, and bold marks the best method per column. *Reference* is OR-Tools GLS for TSP/CVRP and online Best-Fit for BPP.

	TSP			CVRP			BPP		
Method	n50	n100	n200	n50	n100	n200	OR2	OR3	OR4
UCE (ours)	**5.804**	**7.835**	**10.841**	**10.648**	**12.037**	**14.795**	**0.0552**	0.0504	0.0429
FunSearch	5.805	7.880	11.057	11.774	13.993	16.607	0.0586	0.0529	0.0431
HiFo-Prompt	5.935	7.889	10.920	11.147	14.113	15.438	0.0576	**0.0489**	**0.0409**
EoH	6.122	7.913	11.303	11.787	13.478	15.354	0.0621	0.0512	0.0443
MCTS-AHD	5.930	7.935	11.000	11.833	13.727	15.266	0.0556	0.0563	0.0445
ReEvo	6.109	8.085	12.617	12.069	13.391	15.221	0.0581	0.0554	0.0438
Reference	*5.833*	*7.984*	*11.144*	*10.828*	*12.522*	*15.465*	*0.0656*	*0.0566*	*0.0508*

Table 3: From-scratch comparison under three backbones at the training scale. Lower is better, and bold marks the best per column. A dash denotes no usable run.

	CC 4.7			Codex 5.2			Qwen3.8-27B		
Method	TSP-20	CVRP-20	BPP	TSP-20	CVRP-20	BPP	TSP-20	CVRP-20	BPP
UCE (ours)	**3.955**	**6.263**	0.0433	3.987	**6.231**	**0.0431**	**4.169**	**6.316**	**0.0433**
FunSearch	3.956	6.698	0.0479	4.011	6.324	0.0455	4.502	6.511	0.0434
ReEvo	4.047	6.386	0.0452	**3.977**	6.401	0.0433	4.647	6.347	0.0520
EoH	4.299	6.370	**0.0422**	5.294	6.244	0.0460	4.410	6.474	0.0485

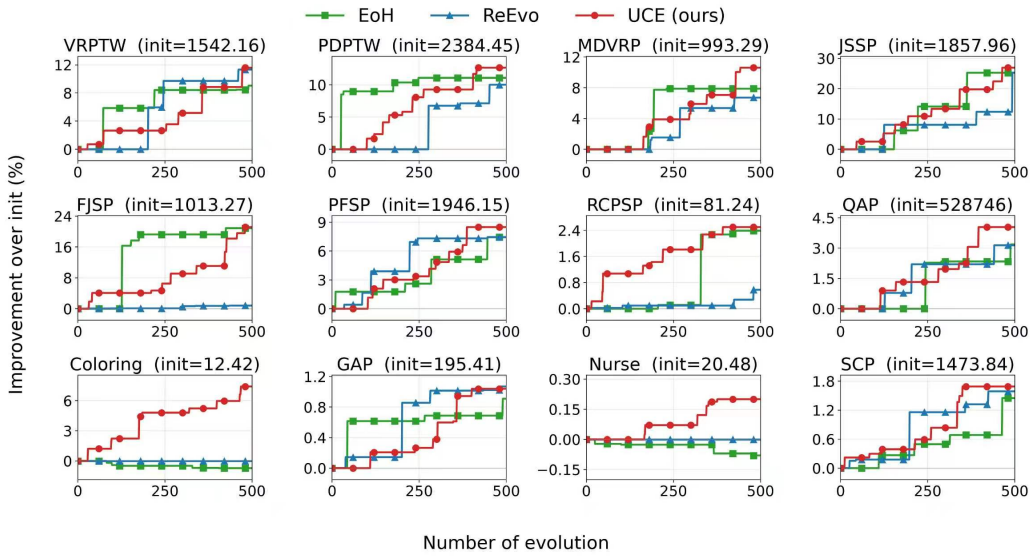

Figure 3: Search progress on the twelve-task OR suite. Each panel plots improvement over the OR-Tools incumbent against the number of evolution steps, with the initial objective given in the title. UCE continues to improve after an early plateau on several tasks. The relative ordering of methods changes during search, so early performance does not always predict the final result.

4.3 Heuristic design from scratch

With the closed-source backbone, UCE ranks first in 7 of 9 settings and second in the other two (Table 2). Programs evolved at $n = 20$ achieve the lowest objective at all three larger test sizes on TSP and CVRP, supporting transfer beyond the training size. On TSP, the margin over the strongest baseline increases with instance size. On BPP, UCE leads on OR2, while HiFo-Prompt achieves the best results on OR3 and OR4.

Across three backbones at the training scale, UCE also ranks first in 7 of 9 settings and second in the other two (Table 3). It achieves the lowest CVRP-20 objective with each backbone, supporting its applicability across the generators evaluated in this comparison.

4.4 Selection errors under replay

To assess selection behavior, we replay the searches and compare each decision with outcomes observed later in the replay. Even when the original search committed or rejected a branch, the replay continues it for a prespecified horizon or gives it additional attempts under the same rule. These counterfactual continuations do not change the programs selected for the main test results. Premature acceptance occurs when an accepted direction is outperformed by a later alternative. Premature rejection occurs when an abandoned direction subsequently yields a better program from the same lineage. These labels describe outcomes within the observed replay horizon; they do not establish what a direction could achieve with further search beyond that horizon.

Table 4: Replay diagnostics. FP and FN are premature acceptance and rejection rates (%). Pearson correlates selection scores with later gains.

Method	FP	FN	Pearson
UCE (ours)	**4.6**	**6.3**	**0.77**
MCTS-AHD	5.2	7.9	0.73
HiFo-Prompt	6.9	8.9	0.69
ReEvo	7.2	11.8	0.54
FunSearch	8.8	7.5	0.55
EoH	9.1	8.3	0.61

UCE has premature acceptance and rejection rates of 4.6% and 6.3%, respectively, compared with 5.2% and 7.9% for MCTS-AHD, the closest baseline (Table 4). Its score at the time of selection also has the highest correlation with subsequent improvement, at 0.77 compared with 0.73 for MCTS-AHD. These results support using evaluations from a common parent to inform selection, with fewer accepted directions later surpassed and fewer rejected directions that later improve. We next examine the contribution of each calibration component.

4.5 Contribution of the calibration components

Table 5: Qwen3.8 ablations on CVRP-20 and TSP-20; mean ± SD across five seeds.

Variant	cvrp20	tsp20
Full UCE	$\mathbf{6.316 \pm 0.078}$	$\mathbf{4.196 \pm 0.047}$
w/o sigma calibration	6.683 ± 0.115	4.473 ± 0.112
w/o mean calibration	6.776 ± 0.253	4.485 ± 0.176
w/o holdout	6.436 ± 0.060	4.222 ± 0.032

We remove each component separately on CVRP-20 and TSP-20 (Table 5). Full UCE achieves the lowest mean objective on both tasks. Removing mean calibration causes the largest degradation among the calibration ablations, and removing dispersion calibration also degrades performance on both tasks. Omitting the holdout step produces a smaller degradation, but remains worse than full UCE on both tasks. These results support using both stage moments for score correction and an independent holdout check for selected high-scoring programs.

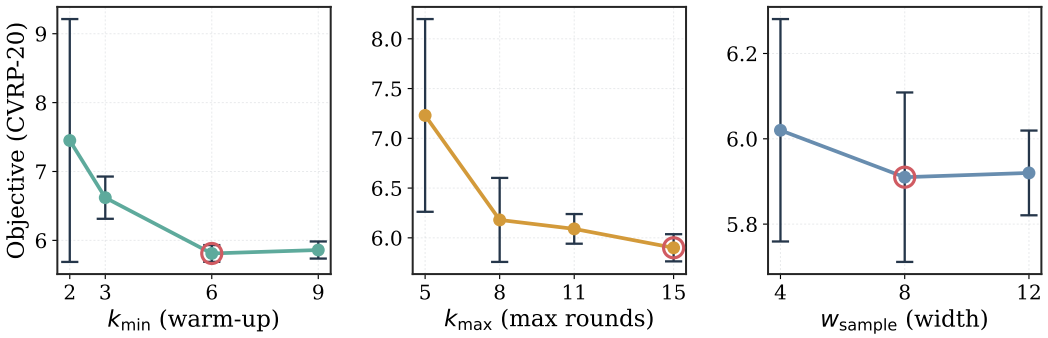

Figure 4: Sensitivity on CVRP-20 to the calibration warm-up $k_{\min}$, the maximum number of rounds $k_{\max}$, and the sampling width w_{sample}. Means over five seeds with 95% confidence intervals. The circled point is the default setting.

4.6 Sensitivity to search parameters

On CVRP-20, performance is most sensitive to the calibration warm-up $k_{\min}$ among the three parameters tested (Figure 4). At $k_{\min} = 2$, the 95% confidence interval spans more than two objective units. Increasing $k_{\min}$ to six reduces the mean objective from 7.45 to 5.81 and substantially narrows the interval; increasing it further to nine provides little additional improvement. This supports a longer warm-up, with diminishing returns beyond six in this experiment. Increasing the maximum number of rounds $k_{\max}$ beyond eight and the sampling width w_{sample} beyond eight changes the mean by less than 0.3 and mainly narrows the interval. The observed sensitivity is specific to the tested task and parameter ranges. We use $k_{\min} = 6$, $k_{\max} = 15$, and $w_{\text{sample}} = 8$ throughout.

5 Conclusion

LLM-AHD uses current program evaluations to guide iterative design, but a single score does not show whether further refinement is worthwhile. UCE compares each candidate with the mean and dispersion of scores from the same parent, then uses the adjusted score for selection and feedback. Across twelve operations-research tasks and nine from-scratch settings, UCE improves overall performance and reduces premature acceptance and rejection relative to the compared methods. Because its reference is limited to sampled programs, it cannot determine whether a poor score reflects the proposed change or its implementation. Execution feedback may help separate these effects.

AI USE STATEMENT

We used generative AI tools to assist with code debugging, grammatical editing of the manuscript, and the generation of intermediate figures and statistical summaries during the research process. The research idea was developed independently by the authors. The authors interpreted the experimental results and formulated the conclusions themselves. The authors take responsibility for the final content of this paper, including AI-assisted materials.

ETHICS STATEMENT

This study uses operations-research benchmark instances and does not involve human subjects, personal data, or sensitive interventions.

REPRODUCIBILITY STATEMENT

We describe UCE and the experimental setup in the main paper and provide further implementation and evaluation details in the appendix. Where open-source implementations are available, our baselines follow the corresponding public projects, with adaptations for method integration, the local environment, and model API interfaces. We plan to provide UCE and adapted baseline code in an anonymous repository when permitted by the authors' internal review and approval process; this submission does not claim that the code is publicly available at submission time. Because the experiments rely in part on externally hosted model APIs, exact reproduction of individual outputs and numerical results may depend on API access and model behavior at the time of execution.

REFERENCES

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

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025. URL https://arxiv.org/abs/2507.19457.

Anastasios N Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 16(4):494–591, 2023.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002. doi: 10.1023/A:1013689704352.

J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

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

Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. QAPLIB–a quadratic assignment problem library. *European Journal of Operational Research*, 102(3):667–673, 1997.

Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. Nurse rostering—a multicriteria approach. *European Journal of Operational Research*, 247(2):676–686, 2015.

Chentong Chen, Mengyuan Zhong, Jialong Shi, Jianyong Sun, and Ye Fan. HiFo-Prompt: Prompting with hindsight and foresight for LLM-based automatic heuristic design. In *Proceedings of ICLR*, 2026.

Jean-Francois Cordeau, Gilbert Laporte, and Anne Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 48 (9):928–936, 1997.

Isaac Gibbs and Emmanuel Candes. Adaptive conformal inference under distribution shift. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Google. OR-Tools: Routing. https://developers.google.com/optimization/routing, n.d. Accessed September 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *Proceedings of ICLR*, 2024. URL https://arxiv.org/abs/2305.11738.

Minh Hieu Ha, Hung Phan, Tung Duy Doan, Tung Dao, Dao Tran, and Huynh Thi Thanh Binh. Pareto-grid-guided large language models for fast and high-quality heuristics design in multi-objective combinatorial optimization. In *Proceedings of AAAI*, 2026.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398. URL https://doi.org/10.1162/106365601750190398.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. URL https://arxiv.org/abs/2408.08435.

Ziyao Huang, Weiwei Wu, Kui Wu, Jianping Wang, and Wei-Bin Lee. Calm: Co-evolution of algorithms and language model for automatic heuristic design. In *Proceedings of ICLR*, 2026.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. Ale-bench: A benchmark for long-horizon objective-driven algorithm engineering. In *NeurIPS Datasets and Benchmarks Track*, 2025.

David S. Johnson and Michael A. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. American Mathematical Society, 1996.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Rainer Kolisch and Arno Sprecher. PSPLIB–a project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, 1997.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025. URL https://arxiv.org/abs/2509.19349.

Haibing Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(2):149–164, 2003.

Xijun Li, Jiexiang Yang, Jinghao Wang, Bo Peng, Jianguo Yao, and Haibing Guan. Strcmp: Integrating graph structural priors with language models for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2025.

Haoran Ling, Yuecheng Li, Zeyu Song, Jing Yao, Shuwen Kang, Chi Lu, Wenjin Wu, and Peng Jiang. RecHarness: A bandit-routed agentic harness for self-evolving recommender systems. *arXiv preprint arXiv:2607.29241*, 2026. URL https://arxiv.org/abs/2607.29241.

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 ICML*, 2024.

Fei Liu et al. Eoh-s: Evolution of heuristic set using large language models for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Huan Ma, Jingdong Chen, Joey Tianyi Zhou, Guangyu Wang, and Changqing Zhang. Estimating LLM uncertainty with evidence. *arXiv preprint arXiv:2502.00290*, 2025. URL https://arxiv.org/abs/2502.00290.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.17651.

Bernardino Romera-Paredes et al. Mathematical discoveries from program search with large language models. *Nature*, 2024. doi: 10.1038/s41586-023-06924-6.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.11366.

Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

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

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. doi: 10.1093/biomet/25.3-4.285.

Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. Conformal prediction under covariate shift. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Jianing Wang, Yang Zhou, Xiaocheng Zhang, Mengjiao Bao, and Peng Yan. Self-evolutionary large language models through uncertainty-enhanced preference optimization. *arXiv preprint arXiv:2409.11212*, 2024. URL https://arxiv.org/abs/2409.11212.

Sixue Xing, Haoyu He, Kerui Wu, Zhuo Yang, Haozheng Luo, Tianfan Fu, and Aarthy Nagarajan. Compute allocation for self-evolving LLMs: From depth-breadth to multi-armed bandits. *arXiv preprint arXiv:2605.29268*, 2026. URL https://arxiv.org/abs/2605.29268.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. In *Proceedings of AAAI*, 2025.

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. In *Advances in Neural Information Processing Systems*, 2024.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025a. URL https://arxiv.org/abs/2505.22954.

Qingyang Zhang, Haitao Wu, Changqing Zhang, Peilin Zhao, and Yatao Bian. Right question is already half the answer: Fully unsupervised LLM reasoning incentivization. *arXiv preprint arXiv:2504.05812*, 2025b. URL https://arxiv.org/abs/2504.05812.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025c. URL https://arxiv.org/abs/2510.04618.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. In *Proceedings of ICML*, 2025.

A Estimating the Local Score Distribution

The observable score pool. For the candidates evaluated from a fixed parent, define $\mathcal{O}_{t,m} = \{(j,\ell) : x_{t,j,\ell} \text{ has been evaluated by round } m\}$ and $N_{t,m} = |\mathcal{O}_{t,m}|$. Equation 3 uses $N_{t,m} = km$ for complete batches. If trajectories contribute unequal numbers of observations, the corresponding empirical moments are

$$\widehat{\mu}_{t,m} = \frac{1}{N_{t,m}} \sum_{(j,\ell)\in\mathcal{O}_{t,m}} x_{t,j,\ell}, \qquad \widehat{\sigma}^2_{t,m} = \frac{1}{N_{t,m}} \sum_{(j,\ell)\in\mathcal{O}_{t,m}} (x_{t,j,\ell} - \widehat{\mu}_{t,m})^2. \tag{7}$$

These statistics describe the observed pool, including low and zero scores when the evaluator assigns them to candidate programs. If some trajectories receive more evaluations than others, they contribute more observations to the pool. The resulting moments reflect this sampling allocation, rather than a uniform average over all possible ideas or programs.

Online updates. To update the moments without storing the full score history, keep the count n, mean M_n, and centered sum of squares Q_n. On receiving raw score x_{n+1}, update

$$\begin{aligned} d_{n+1} &= x_{n+1} - M_n, \\ M_{n+1} &= M_n + \frac{d_{n+1}}{n+1}, \\ Q_{n+1} &= Q_n + d_{n+1}(x_{n+1} - M_{n+1}). \end{aligned} \tag{8}$$

Initialize $M_1 = x_1$ and $Q_1 = 0$. The empirical variance is Q_n/n, which differs from the unbiased sample-variance estimate $Q_n/(n-1)$. Only raw evaluator scores enter these statistical updates.

Gaussian approximation and sampling uncertainty. Equation 4 defines a Gaussian reference using the pool's mean and dispersion. Its standard deviation $\widehat{\sigma}_{t,m}$ measures variation among observed implementations. The width of an interval for the unknown mean is a different quantity. Under independent, identically distributed observations with finite variance the central limit theorem gives

$$\frac{\sqrt{N}(\bar{X}_N - \mu)}{\sigma} \xrightarrow{d} \mathcal{N}(0,1). \tag{9}$$

A Bernoulli event model is the special case of estimating an event probability, where $\sqrt{N}(\widehat{p}_N - p)/\sqrt{p(1-p)}$ has the corresponding normal limit when $0 < p < 1$. This does not imply that raw program scores are Gaussian or that a Bernoulli variable is equivalent to a Gaussian variable.

Within-stage feedback can make successive observations dependent and change their distribution. Equation 1 does not remove that dependence. The Gaussian interval in the main text therefore serves as a descriptive score reference. It does not provide nominal confidence coverage by treating all km evaluations as independent. A larger pool can provide a more stable description of the sampled scores, but individual programs may continue to differ substantially in performance.

B Score Calibration and Loop Integration

Rank-dependent weights. To express the dependence on the existing search procedure, let $r_{t,j,m}$ denote proposal stream j's standing in its existing k-sample ranking. Write the low-score penalty weight abstractly as

$$b_{t,j,m} = \mathcal{W}(r_{t,j,m}, z_{t,j,m}, \mathcal{H}_{t,m}), \qquad b_{t,j,m} \geq 0. \tag{10}$$

The function $\mathcal{W}$ determines the strength of the low-score adjustment from the proposal-stream rank, the standardized score, and the accumulated history. Equation 10 states these inputs without imposing a particular numerical weighting rule. Choosing $\mathcal{W}$ determines how the score adjustment responds to the available evidence.

Effect on an individual score. Subtracting $x_{t,j,m}$ from Equation 6 gives the correction in the original score units,

$$\widetilde{x}_{t,j,m} - x_{t,j,m} = (1 - b_{t,j,m})(\mu - x_{t,j,m})_+. \tag{11}$$

Penalty weights below one reduce the downward departure of below-center scores, whereas scores at or above the stage mean are unchanged and are checked through the independent holdout audit. The scale σ enters the weighting map through the standardized departure $z_{t,j,m}$. With weights fixed independently of z, it cancels algebraically from the score correction. This makes explicit where dispersion must affect an instantiation of $\mathcal{W}$.

For fixed ranking and bounded weights continuous in z, the transformed score is continuous at the reference mean and equals the raw score for $z \geq 0$. Setting $b = 1$ gives $\widetilde{x} = x$ below the mean as well. These properties concern the score transformation for a given reference and ranking, and they do not imply that discrete ranking changes themselves are continuous.

Order of operations. To keep the reference tied to evaluated performance, each round adds the new raw scores to the observation pool and updates its moments. The agent obtains proposal-stream ranks from its original procedure, computes standardized departures and calibrated scores, and uses those scores in the next iteration. Calibrated scores are never fed back as additional raw observations. Before the warm-up requirement is met, the original scores provide feedback. A positive scale floor handles a pool with no observed dispersion.

Optional transition and pruning rules. Score calibration does not require an acceptance or rejection threshold. The surrounding search procedure may use calibrated scores to decide when to replace the parent or stop sampling a proposal stream. It may also require validation on separate instances or repeated evidence of improvement before accepting a candidate. Such rules determine how calibrated scores guide search. After warm-up, the adjustment is defined for every evaluated candidate.

C Experimental Details

Benchmarks. **TSP** evolves on 128 TSP20 instances with coordinates drawn uniformly from $[0, 1]^2$ and disjoint calibration, audit, and frozen test sets of 32 instances each, and the metric is mean tour length. **CVRP** follows the same construction with 20 customers, uniform demands, and a fixed vehicle capacity, and the metric is mean route length. Both transfer to 1000 fresh instances at each of $n \in \{50, 100, 200\}$, against OR-Tools Guided Local Search (Google, n.d.) at five seconds per instance. **BPP** evolves on OR-Library `binpack1` (Beasley, 1990) ($C = 150$, with ten fit, five calibration, and five audit instances) and is scored by the mean excess ratio $(B_j - lb_j)/lb_j$ over the LP bound $lb_j = \sum_i s_{j,i}/C$. Final evaluation uses `binpack2/3/4` (OR2–OR4) against online Best-Fit. The protocol uses three independent training searches and evaluates the best training program on a held-out split. The common call allowance and method-specific search controls are detailed in Appendix D.

Twelve-task suite. Tasks and public instances follow QAPLIB (Burkard et al., 1997), DIMACS coloring (Johnson & Trick, 1996), Solomon VRPTW (Solomon, 1987), Li–Lim PDPTW (Li & Lim, 2003), Cordeau MDVRP (Cordeau et al., 1997), classical JSSP and FJSP (Adams et al., 1988; Brandimarte, 1993), Taillard PFSP (Taillard, 1993), PSPLIB RCPSP (Kolisch & Sprecher, 1997), OR-Library GAP and SCP (Beasley, 1990), and the INRC nurse-rostering benchmark (Ceschia et al., 2015). OR-Tools provides the initialization incumbent on the train-64 split, and each method's best program is re-run on an independent 128-instance test pool per task.

D Baseline Reproduction Settings

The following settings specify the baseline reproductions used in our experiments. Each method retains its own search controls under the common allowance of 500 LLM calls per search, or at most 600 including debugging.

FunSearch. Our FunSearch-like reproduction uses four islands, a population size of five, and two elites. Its native iteration limit is 500.

EoH. The adapter uses the four operators `e1`, `e2`, `m1`, and `m2`, and a population size of ten. Its native sample limit is 500.

ReEvo. We use a population size of ten, five reflection pairs, and two mutants. Reflection calls count toward the common LLM-call allowance.

MCTS-AHD. We use an initial population of four, a population size of ten, and a native limit of 500 function evaluations.

HiFo-Prompt. We set `ec_pop_size=8` and `ec_n_pop=8`.

Budget accounting. Native iteration, sample, and evaluation limits do not replace the shared LLM-call allowance: generation and reflection requests count toward the 500-call search limit, with debugging included in the 600-call maximum. Three independent training searches are performed per method; the program with the best training objective is evaluated on the frozen test split. Counterfactual continuations are accounted for separately from online search.