# RADAR: RUNTIME-AWARE DIAGNOSIS AND RACING FOR AGENTIC SOLVER DESIGN

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

LLM-based automated algorithm design often uses a solver's final scalar score both to rank candidates and to guide code revisions. Yet similar scores can conceal different runtime behavior that calls for different edits, a problem we term *behavioral aliasing*. In controlled studies of MCTS-AHD and ReEvo, we find no detectable performance gain from exposing exact score magnitudes or using run-specific reflections. We introduce **RADAR** (Runtime-Aware Diagnosis and Racing), an agentic framework that separates the two roles: execution traces guide revisions, and scores allocate evaluation effort. A trusted evaluator records when valid solutions improve, how long the solver runs without further improvement, and why outputs are rejected; a diagnostic agent turns this trace and the search history into one measured bottleneck and one code edit for a writer agent. Racing first runs each new program on a single training sample and stops weak programs before they are tested on the remaining samples, and Branch-Adaptive Reflection changes the edit strategy when a search branch stalls. Across seven benchmarks, RADAR with Qwen3-32B achieves the best mean score among the evaluated search baselines on every task; on CVRP-200 its gap to the reference is 7.20% versus 11.76% for the strongest baseline. With GPT-5.4-mini, RADAR leads or ties on five of the seven tasks. Attaching our racing and diagnosis mechanisms to MCTS-AHD and ReEvo improves both methods on all three tested tasks.

## 1 INTRODUCTION

Combinatorial Optimization Problems (COPs) often require good solutions under tight time limits. As instances grow, exhaustive search becomes impractical, and solver quality depends on heuristics that take domain expertise and repeated testing to develop. The resulting solver code is tailored to the problem at hand, so experts often have to redesign and retune it for a different problem.

Large Language Model (LLM)-based Automated Heuristic Design (AHD) offers a way to reduce this repeated expert effort. In an agentic AHD loop, an LLM proposes solver code, an evaluator runs it, and a search controller uses the result to guide the next proposal. Existing systems organize this loop

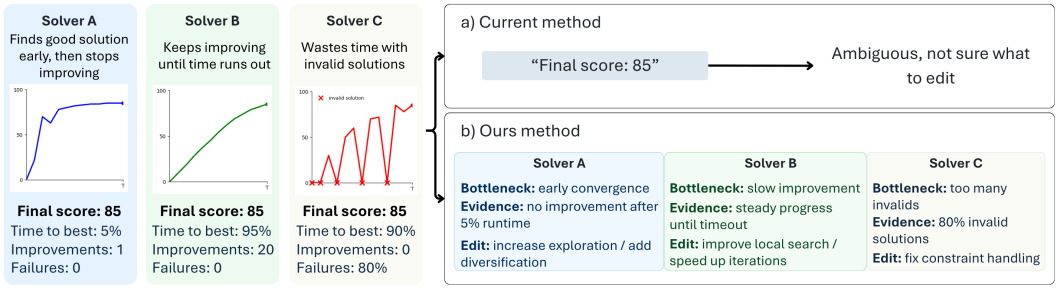



Figure 1: Same score, different behavior. (a) Three solvers reach the same final score via early stagnation, improving-at-timeout, and invalid-heavy search, leaving their next edits ambiguous. (b) RADAR combines the score with execution traces and search history to distinguish these behaviors and produce more specific revision guidance.



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.

differently: FunSearch (Romera-Paredes et al., 2024) and EoH (Liu et al., 2024) use evolutionary search, ReEvo (Ye et al., 2024) adds written reflections, and MCTS-AHD (Zheng et al., 2025) and MOTIF (Kiet et al., 2026) organize candidates in a search tree. Most of these systems search over heuristic components within a fixed solver, while recent work on Automated Algorithm Design (AAD) extends the search to complete solver programs (Chen et al., 2026a).

Despite these differences, feedback for the next decision is often dominated by a program's final objective score, such as route length or collected prize. The score serves two roles: *allocation*, deciding where to spend search and evaluation effort, and *direction*, deciding how to revise a selected program. Scores provide a useful ranking for allocation, but they reveal little about the runtime behavior that a revision should address.

Fig. 1 illustrates the ambiguity. Solvers $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ reach the same final score, yet $\mathbf{A}$ finds its best solution early and stalls, $\mathbf{B}$ continues improving until the time limit, and $\mathbf{C}$ produces many invalid outputs. Their next edits should differ, although their final scores cannot distinguish them. This *behavioral aliasing* makes a final score an ambiguous guide to the next code edit. Work on anytime algorithms likewise characterizes a solver by its solution quality over time rather than by its final result alone (Dean & Boddy, 1988; Zilberstein, 1996; López-Ibáñez & Stützle, 2014). DASH (Wang et al., 2026) makes a similar observation about convergence trajectories and responds by adding a convergence-rate term to how solvers are ranked. We draw a different conclusion: when the goal is the quality reached at the time limit, the final score remains the right criterion for ranking, and the ambiguity matters for revision, because the score cannot say which edit a program needs. To test whether the existing AHD feedback resolves this ambiguity, we examine two signals in controlled studies. In MCTS-AHD, exact parent-score magnitudes yield no detectable gain once program ordering is preserved (Section 4.2), and in ReEvo, replacing a run's written reflections with reflections from an unrelated run likewise has no detectable effect (Section 4.2). This points to execution behavior as a source of the missing revision guidance.

We build **Runtime-Aware Diagnosis and Racing (RADAR)** around this split: execution traces guide revisions, and scores allocate evaluation effort (Fig. 2). A trusted evaluator records each program's *execution trace*: when valid solutions improve, how long the solver runs without further improvement, and why outputs are rejected. A diagnostic agent combines this trace with search history to identify a measured bottleneck and specify one code edit for a writer agent. For allocation, *racing* tests each new program on one anchor training sample and promotes only those that pass a score gate to the remaining samples. When a search branch stalls, *Branch-Adaptive Reflection* (BAR) changes the requested edit from Refine to Repair or Rethink. Our contributions are as follows:

- **Separating ranking from revision.** We argue that behavioral aliasing is a revision problem rather than a ranking problem, and support both halves: in our benchmarks the endpoint score ranks programs reliably (one anchor run keeps 99.2% of the eventual top-10% programs; Section 4.5), while exact score magnitudes and score-derived reflections add no detectable revision value (Section 4.2) and the parent's own execution trace does, beating withheld or mismatched traces on all three tested tasks (Section 4.4).
- **RADAR.** We introduce an agentic framework that turns trusted execution traces into one measured bottleneck and one bounded edit for a writer agent. Scores keep a separate role in racing, and Branch-Adaptive Reflection adjusts the edit strategy when a search branch stalls.
- **Effectiveness and portability.** Under equal solver-run budgets with Qwen3-32B, RADAR achieves the best mean among the evaluated methods on all seven benchmarks, e.g., a 7.20% gap to the reference on CVRP-200 versus 11.76% for the strongest baseline, and with GPT-5.4-mini it leads or ties on five of seven. Plugging RADAR's racing and diagnosis into MCTS-AHD and ReEvo also improves both on all three tested tasks (Section 4.6), and with Qwen3-32B RADAR closes 79–99% of the gap between MCTS-AHD run with Qwen3-32B and with GPT-5.4 on OP-200, CVRP-200, and UFLP-1000 (Appendix L).

## 2 Related Work

FunSearch (Romera-Paredes et al., 2024), EoH (Liu et al., 2024), ReEvo (Ye et al., 2024), MCTS-AHD (Zheng et al., 2025), and MOTIF (Kiet et al., 2026) are LLM-based AHD methods that edit one heuristic component inside a fixed solver skeleton, while $A_2$DEPT (Chen et al., 2026a) is an

automated algorithm design (AAD) method that edits complete solver programs. Besides what they edit, most of these search loops focus on how to choose which program to revise, while the writer still revises from the parent's code and final score, or from text derived from scores. Consistent with this, recent analyses find simple baselines competitive with these search loops (Gideoni et al., 2026) and gains concentrated in a few types of edits (Pelleriti et al., 2026), suggesting that the revision signal, rather than the choice of which program to revise, is the weak point.

One way to enrich the revision signal is self-reflection (Shinn et al., 2023; Madaan et al., 2023), as in ReEvo, but LLMs struggle to self-correct without external feedback (Olausson et al., 2024; Huang et al., 2024b). Execution-guided methods supply that feedback through execution results (Chen et al., 2024), runtime traces (Haque et al., 2025; Cheng et al., 2024), or profiling (Shypula et al., 2024; Huang et al., 2024a; Lei et al., 2025), but they rely on a clear failure signal, such as a failing test or a slow function. An anytime heuristic usually has none: it runs to the time limit and returns a valid but mediocre answer, and what needs fixing is visible only in how its solution quality evolves and which of its outputs are rejected. Existing AHD feedback does not capture this: LaGO (Yu et al., 2026) and teacher-aware evolution (Chen et al., 2026b) describe the solutions a program produces rather than how it behaved while running, and DASH (Wang et al., 2026) summarizes convergence trajectories in a metric that improves the choice of which solver to evolve rather than how to revise its code.

Better revision signals help only if the search can afford to try many revisions, yet most LLM-based AHD methods, including our baselines, evaluate every program on all training samples, spending as much on a program that crashes as on a promising one. Hercules-P (Wu et al., 2025) and NeRM (Guo et al., 2025) reduce this cost by predicting performance instead of measuring it: Hercules-P has an LLM predict a new heuristic's fitness from its code similarity to evaluated heuristics, and NeRM discards candidates that a learned pairwise predictor ranks as weak. A misprediction either keeps a weak program, so the search wastes budget revising it, or discards a strong one untested, and both papers' own ablations show that the saved time costs solution quality. Outside AHD, racing and successive halving spend little budget on weak candidates in algorithm configuration (Birattari et al., 2002; Jamieson & Talwalkar, 2016; Li et al., 2018), but over a fixed set of candidates rather than a stream of new programs; AlphaEvolve uses cascades of tests whose thresholds the user sets for each task (Novikov et al., 2025), and ShinkaEvolve rejects near-duplicate proposals before evaluation (Lange et al., 2026), which removes redundant programs but not poor ones.

To bridge these gaps, RADAR keeps a standard tree search ranked by endpoint scores, turns each parent's execution trace into one bounded edit, and measures every new program on one real training sample instead of predicting its fitness.

## 3 Method: Runtime-Aware Diagnosis and Racing (RADAR)

RADAR uses scores to decide which candidates merit further evaluation and execution traces to guide their revision. Figure 2 shows an iteration in which the selected expansion revises one parent. In (a), the controller selects a parent program $p$, and Branch-Adaptive Reflection (BAR) chooses how to approach its revision. In (b), the diagnostic agent uses $p$'s trace to specify an edit, which the writer agent implements as a child program $p'$. In (c), the evaluator runs $p'$ and records its behavior; racing decides whether to evaluate it on the remaining training samples. Promoted children return to the search tree with their scores and traces, supplying feedback for the next iteration.

### 3.1 Evaluator Feedback: Scores and Execution Traces

For each program $p$, the evaluator records an *execution trace* $\tau(p)$ containing scores and runtime behavior. Its *anchor score* $u_0(p)$ comes from one fixed training sample; if racing promotes $p$ (Section 3.5), its *training score* $f(p)$ is the mean over all five samples. Racing uses the scores to allocate evaluation, while diagnosis uses the runtime record to guide code revisions.

**Trusted scoring.** Each generated program runs in a separate process with a wall-clock limit. The prompt forbids file, network, and GPU access, although no OS-level sandbox enforces these restrictions. The program reports proposed improvements through a fixed function call; a separate evaluator timestamps each report, checks feasibility, and computes its score.

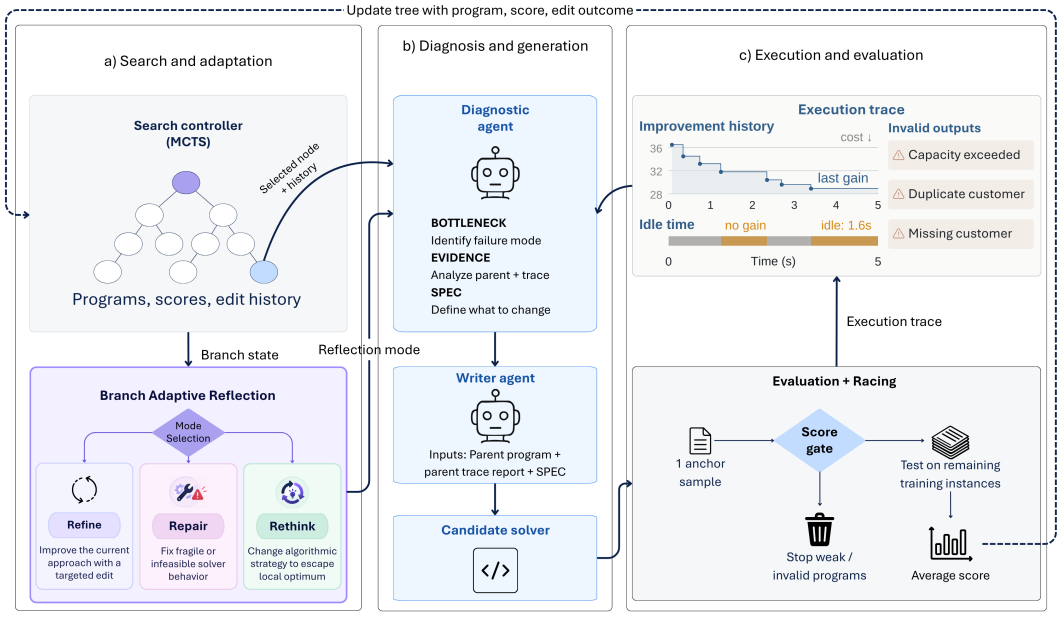



Figure 2: RADAR framework. **(a)** The MCTS controller selects a parent program, and BAR chooses a reflection mode. **(b)** The diagnostic agent turns the parent's trace and search history into an edit specification, which the writer implements as a child program $p'$. **(c)** The evaluator runs $p'$ on an anchor sample; racing promotes candidates that pass the score gate to the remaining training samples. Promoted children return their scores and traces to the search tree (dashed loop).

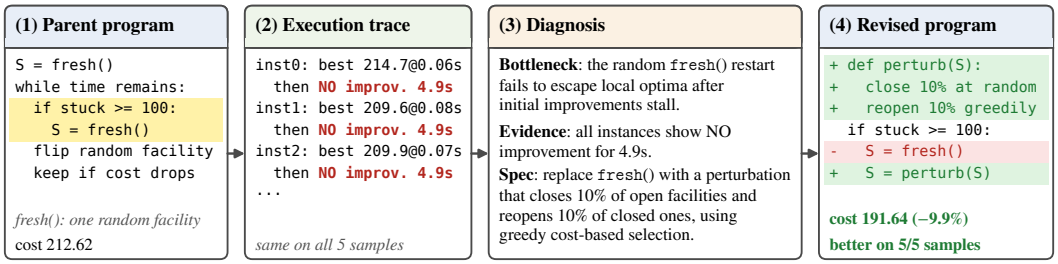



Figure 3: One diagnosis step on UFLP-1000 (code abstracted). The parent stops improving within 0.1 s on every training sample; the diagnosis replaces its random restart with a perturbation of the current solution, lowering cost by 9.9%.

**What the execution trace contains.** Besides scores, the evaluator records three kinds of runtime information for each training sample (Fig. 2). The **improvement history** $\{(t_k, z_k)\}_{k=1}^{K}$ records the time and score of each improved valid solution. When a valid solution exists, its last improvement gives the **idle time** $\Delta t_{\text{idle}} = T_{\text{budget}} - t_K$. The **invalid-output summary** counts reported and valid solutions and summarizes rejection reasons. The diagnoser receives these fields as a one-line report per sample (Appendix A.6). They distinguish the behaviors in Fig. 1: early stagnation (**A**), continued improvement (**B**), and repeated invalid outputs (**C**). The trace adds under 2% runtime.

### 3.2 Search Controller: Parent Selection and Tree Update

RADAR's main controller is a Monte Carlo tree search (MCTS), in which each promoted program is a node linked to its parent (Fig. 2(a)). To select a parent, the controller descends from the root by the UCT rule, choosing at each level the child with the highest normalized training score plus an exploration bonus that decays over time. It expands the selected node using an MCTS-AHD operator that asks the writer to modify one parent, combine several programs, or propose a different design

(Appendix A.3). A promoted child enters the tree with its training score, trace, and edit outcome; ancestors take the best descendant score. Programs stopped by racing do not enter the tree.

**Search history for diagnosis.** The tree also retains an *edit history*: for each earlier diagnosis, the edit it requested and the *edit outcome*, i.e. the resulting score change. The diagnoser sees the six most recent programs and the best program of each of up to eight algorithm families, each with its score, together with the outcomes of its last five edits (Appendix A.6); this helps it avoid repeating failed edits.

### 3.3 Branch-Adaptive Reflection

Search can stall within a family of similar programs: the branch best stops improving, yet subsequent edits continue to refine the same design. A *branch* is a founder program and all its descendants in the search tree. BAR measures stagnation with a *Branch Stagnation Index* $S = 0.65A + 0.35F \in [0, 1]$, counting only clean evaluations (feasible programs fully tested on five samples and terminated normally). Here $A$ counts evaluations since the branch best last improved, scaled to reach 1 after 12; $F$ is the fraction of the four most recent evaluations that did not improve on their parent. A branch counts as stalled when $S \geq 0.70$ and additional safeguards hold (Appendix A.5).

BAR selects the next diagnosis mode (Fig. 2(a)): **Refine** by default, **Repair** for a stalled branch if recent edits are infeasible or the search-wide best still improves, and **Rethink** otherwise. Repair requests a change to a constant, threshold, or schedule; Rethink requests replacement of one construction, local-search, or metaheuristic component. Each switch lasts one diagnosis (Appendix I).

### 3.4 Diagnosis-Guided Program Revision

The diagnostic agent turns a parent's measured behavior and search history into a request for one code edit. It reads the parent program, its trace report, and the program, score, and edit history, then returns three lines: (1) Bottleneck, the main observed problem, such as early stagnation or repeated invalid outputs; (2) Spec, one concrete edit naming the function or data structure to change; and (3) Evidence, a measurement that supports the proposed edit. The writer agent receives the parent program, the same trace report, and the Spec, and implements the edit as a complete child program $p'$ (Fig. 3). RADAR requests a diagnosis when the selected MCTS-AHD operator revises one parent (e2, m1, or m2); other expansion operators retain their original prompts (Appendix B).

### 3.5 Racing: Allocating Evaluation Budget

Evaluating every candidate on all five training samples spends the same budget on weak and strong programs. RADAR instead uses *racing*: each new program $p$ first runs on one fixed *anchor sample* $x_0$, and its anchor score $u_0(p)$ (oriented so that larger is better) passes through a *score gate*. After a short warm-up, the gate compares $u_0(p)$ with the anchor scores $\mathcal{H}_0$ of earlier programs and keeps a fraction $q = 0.40$: programs in the top 40% are promoted and tested on the remaining four training samples; the rest stop after one solver run. Formally,

$$\textsc{ScoreGate}(p, x_0) = \begin{cases} \text{promote} & u_0(p) \geq Q_{1-q}(\mathcal{H}_0) \\ \text{stop (1 solver run)} & \text{otherwise,} \end{cases} \tag{1}$$

where $Q_{1-q}$ is the empirical $(1-q)$-quantile. Programs that crash or return no valid solution on the anchor sample are always stopped. The warm-up length, anchor choice, and exact tie rules are in Appendix A.4, and Section 4.5 and Appendix G validate the rule empirically.

A promoted child $p'$ enters the tree with its training score $f(p')$, trace, and edit outcome (Fig. 2(c)); its trace can guide later diagnoses. Racing and diagnosis also apply to MCTS-AHD and ReEvo (Section 4.6), whereas BAR requires branch ancestry.

Table 1: Controls on existing AHD feedback: mean ± standard deviation over eight paired searches. (a) MCTS-AHD with parent scores shown or hidden in the revision prompt; two-sided exact paired sign-flip $p$. (b) ReEvo reflection conditions; paired two-sided $t$-test against Original.

**(a)** Score magnitudes (MCTS-AHD)

| Task | Shown | Hidden | $p$ |
|---|---|---|---|
| TSP-50 (↓) | $5.99 \pm 0.05$ | $6.01 \pm 0.08$ | .617 |
| CVRP-50 (↓) | $12.67 \pm 2.52$ | $12.44 \pm 2.18$ | .859 |
| MKP-100 (↑) | $23.02 \pm 0.28$ | $23.13 \pm 0.10$ | .484 |
| OP-50 (↑) | $14.49 \pm 0.69$ | $14.58 \pm 0.39$ | .820 |

**(b)** Written reflection (ReEvo)

| Condition | CVRP-50 (↓) | $p$ | OP-50 (↑) | $p$ |
|---|---|---|---|---|
| Original | $9.89 \pm 0.40$ | – | $14.61 \pm 0.42$ | – |
| Swapped | $9.99 \pm 0.41$ | .562 | $14.87 \pm 0.15$ | .198 |
| Fixed advice | $10.62 \pm 0.32$ | .001 | $14.66 \pm 0.36$ | .815 |
| No reflection | $10.00 \pm 0.59$ | .625 | $14.80 \pm 0.18$ | .268 |

# 4 Experiments

## 4.1 Experimental Setup

The benchmarks: traveling salesperson (TSP), capacitated vehicle routing (CVRP) and its time-window variant (CVRPTW), orienteering (OP), multidimensional knapsack (MKP), uncapacitated facility location (UFLP), and permutation flow shop (PFSP) as listed in Table 4 and Appendix C.

Each search uses five training samples to select a solver, which is then evaluated on 64 fixed held-out samples; test scores do not enter search. An independent search receives 500 solver runs, each executing one program on one training sample for at most five seconds. We run four searches per method and model, using Qwen3-32B (hidden reasoning disabled) and GPT-5.4-mini in the main comparison. Unless stated otherwise, subsequent analyses use Qwen3-32B.

We adapt EoH, ReEvo, MCTS-AHD, and MOTIF to generate complete programs while retaining their search procedures and a shared evaluation interface (Appendix D). Equal solver-run budgets allow RADAR more proposals through racing and do not match LLM calls; Appendix H compares diagnosis with score-only search at similar call counts.

## 4.2 Does Existing AHD Feedback Guide Revision?

**Effect of score magnitudes on program revision.** If behavioral aliasing limits what a final score reveals about the next edit, exposing a parent's score magnitude may provide little additional guidance to the writer. We test this in MCTS-AHD by hiding the numeric parent scores from the revision prompt (*Hidden*) and comparing with the unmodified prompt (*Shown*). The rest of the prompt and search procedure remain unchanged: the search still ranks programs by score and uses this ranking to choose which parents appear in the prompt and in what order, and the prompt still asks for a program with a better score. The two conditions therefore differ only in whether the writer sees the score magnitudes when editing. We run eight paired searches on each of TSP, CVRP, MKP, and OP at size 50–100 (smaller sizes keep the 16-search paired design feasible). No task shows a significant difference (Table 1a; exact paired sign-flip test, all Holm-adjusted $p = 1.0$): beyond the ranking, the score magnitudes give the writer no detectable guidance on what to edit (Appendix E.1).

**Does ReEvo's written reflection contribute to revision?** ReEvo asks a reflector LLM to turn the differences between a lower- and a higher-scoring program into written advice for later programs. On CVRP-50 and OP-50 (eight paired seeds), *Original* uses the current run's reflection, *Swapped* supplies genuine reflection from a different run on the same task at the corresponding iteration, *Fixed advice* repeats one generic recommendation, and *No reflection* removes the advice while keeping the number of evaluated programs. Original is not detectably better than Swapped or No reflection on either task (Table 1b), and No reflection uses 30% fewer LLM calls; only Fixed advice hurts, on CVRP. We therefore find no measurable benefit from run-specific reflection in this control (Appendix E.2).

## 4.3 Main Benchmark Results Across Seven Domains

Table 2 compares RADAR with the adapted MCTS-AHD, EoH, ReEvo, and MOTIF and the simple Best-of-$N$ method, all under the same budget of 500 solver runs, and separately reports the best hand-designed solver with the same five-second runtime. Because some tasks minimize a cost and

Table 2: Main results on 64 held-out test samples per problem (5 s per sample) with **(a)** Qwen3-32B and **(b)** GPT-5.4-mini for all LLM-based methods. †: adapted by us from fixed-skeleton releases to full programs. Classical is the best hand-designed solver under the same 5 s runtime; Reference$*$ is a proven optimum, best-known solution, or bound (Appendix C). Cells show mean score±standard error over four searches, with the gap to the reference below (lower is better). Bold marks the best LLM-based result per panel, including ties.

| | *Non-LLM* | | *LLM-based* | | | | | |
|---|---|---|---|---|---|---|---|---|
| Task | Reference$*$ | Classical (5 s) | Best-of-$N$ | MCTS-AHD† | EoH† | ReEvo† | MOTIF† | **RADAR (Ours)** |
| **(a)** *Qwen3-32B* | | | | | | | | |
| OP-200 | 85.80 | 49.81 | 46.88±0.60 | 57.00±0.30 | 55.61±1.06 | 56.00±0.84 | 57.50±0.30 | **58.91±0.42** |
| (↑) | 0.00% | 41.95% | 45.36% | 33.57% | 35.19% | 34.73% | 32.98% | **31.34%** |
| TSP-200 | 10.82 | 11.30 | 11.38±0.01 | 11.37±0.06 | 11.70±0.24 | 11.40±0.09 | 11.27±0.05 | **11.16±0.05** |
| (↓) | 0.00% | 4.44% | 5.18% | 5.08% | 8.13% | 5.36% | 4.16% | **3.14%** |
| CVRP-200 | 22.37 | 25.44 | 25.00±0.46 | 27.63±0.32 | 26.28±0.45 | 25.58±0.55 | 27.23±0.69 | **23.98±0.19** |
| (↓) | 0.00% | 13.72% | 11.76% | 23.51% | 17.48% | 14.35% | 21.73% | **7.20%** |
| CVRPTW-100 | 13.59 | 14.45 | 16.89±1.04 | 16.85±0.94 | 19.79±0.88 | 18.04±0.32 | 19.19±0.75 | **15.39±0.79** |
| (↓) | 0.00% | 6.33% | 24.28% | 23.99% | 45.62% | 32.74% | 41.21% | **13.25%** |
| MKP-500 | 105.64 | 102.01 | 103.74±0.24 | 104.52±0.33 | 104.49±0.37 | 104.09±0.51 | 103.94±0.20 | **104.54±0.27** |
| (↑) | 0.00% | 3.44% | 1.80% | 1.06% | 1.09% | 1.47% | 1.61% | **1.04%** |
| UFLP-1000 | 184.70 | 186.15 | 187.20±0.94 | 186.84±0.52 | 185.57±0.26 | 188.95±2.15 | 188.06±0.45 | **185.15±0.09** |
| (↓) | 0.00% | 0.79% | 1.35% | 1.16% | 0.47% | 2.30% | 1.82% | **0.24%** |
| PFSP-500 | 25832 | 25836 | 26273±140 | 25965±56 | 25935±24 | 26070±155 | 26329±150 | **25899±12** |
| (↓) | 0.00% | 0.02% | 1.71% | 0.51% | 0.40% | 0.92% | 1.92% | **0.26%** |
| **(b)** *GPT-5.4-mini* | | | | | | | | |
| OP-200 | 85.80 | 49.81 | 54.49±0.51 | 57.99±0.58 | 56.63±0.52 | 56.48±0.76 | 55.76±1.21 | **58.32±0.32** |
| (↑) | 0.00% | 41.95% | 36.49% | 32.41% | 34.00% | 34.17% | 35.01% | **32.03%** |
| TSP-200 | 10.82 | 11.30 | **10.87±0.02** | 10.92±0.04 | 10.91±0.02 | 10.93±0.02 | 11.00±0.08 | **10.87±0.02** |
| (↓) | 0.00% | 4.44% | **0.46%** | 0.92% | 0.83% | 1.02% | 1.66% | **0.46%** |
| CVRP-200 | 22.37 | 25.44 | 23.41±0.04 | 23.46±0.14 | 23.37±0.01 | 23.27±0.03 | 23.90±0.48 | **23.22±0.02** |
| (↓) | 0.00% | 13.72% | 4.65% | 4.87% | 4.47% | 4.02% | 6.84% | **3.80%** |
| CVRPTW-100 | 13.59 | 14.45 | 14.69±0.33 | 14.50±0.10 | 14.47±0.12 | 14.53±0.04 | 16.40±0.53 | **14.08±0.13** |
| (↓) | 0.00% | 6.33% | 8.09% | 6.70% | 6.48% | 6.92% | 20.68% | **3.61%** |
| MKP-500 | 105.64 | 102.01 | **105.54±0.03** | 105.49±0.10 | 105.52±0.05 | 105.53±0.04 | 105.13±0.19 | 105.49±0.06 |
| (↑) | 0.00% | 3.44% | **0.09%** | 0.14% | 0.11% | 0.10% | 0.48% | 0.14% |
| UFLP-1000 | 184.70 | 186.15 | 184.76±0.04 | 184.86±0.06 | 184.88±0.12 | **184.75±0.02** | 185.03±0.24 | 184.79±0.04 |
| (↓) | 0.00% | 0.79% | 0.03% | 0.09% | 0.10% | **0.03%** | 0.18% | 0.05% |
| PFSP-500 | 25832 | 25836 | **25842±3** | **25842±2** | 25900±62 | 25844±3 | 26001±91 | **25842±2** |
| (↓) | 0.00% | 0.02% | **0.04%** | **0.04%** | 0.26% | 0.05% | 0.65% | **0.04%** |

others maximize a value, we also report the percentage gap $\Delta_{\text{opt}} = |\text{Obj} - \text{Opt}|/|\text{Opt}| \times 100\%$ (lower is better), where Opt is a proven optimum (MKP, UFLP), a best-known solution (TSP, CVRP, CVRPTW), or a bound (PFSP, OP); see Appendix C.

The full-program adaptations are consistently stronger than their native fixed-skeleton counterparts (Appendix D, Table 5), so they are the baselines we compare against. With Qwen3-32B, RADAR achieves the best mean result among LLM-based methods on all seven tasks, while its racing tests 210 programs per search on average under the same 500 solver runs. With four searches per method, these comparisons are descriptive: some margins are large relative to the standard errors (e.g., CVRP-200, $23.98 \pm 0.19$ vs. $25.00 \pm 0.46$ for Best-of-$N$), whereas on MKP-500 RADAR leads MCTS-AHD by only 0.02 (Appendix N discusses why gains differ across tasks). With GPT-5.4-mini (Table 2b), RADAR has the lowest gap on OP-200, CVRP-200, and CVRPTW-100 (3.61% vs. 6.48% for the best baseline there) and ties the best method on TSP-200 and PFSP-500; on MKP-500 and UFLP-1000 all methods except MOTIF lie within 0.15% of the reference, and the best baseline leads RADAR by at most 0.05 percentage points. In an additional run with GPT-4.1-mini on OP-200, CVRP-200, and UFLP-1000, RADAR achieves the best mean on all three tasks (Appendix K).

**Across model backbones.** Table 2 compares search methods within each model. Figure 4 extends the comparison across models, using MCTS-AHD with GPT-5.4 as a stronger-model reference

under the same solver-run protocol. Relative to the gap between MCTS-AHD with Qwen3-32B and GPT-5.4, RADAR with Qwen3-32B closes 99% on OP-200, 82% on CVRP-200, and 79% on UFLP-1000. The effect is much smaller on MKP-500, where MCTS-AHD with Qwen3-32B is already close to the task reference. These comparisons are descriptive (Appendix L).

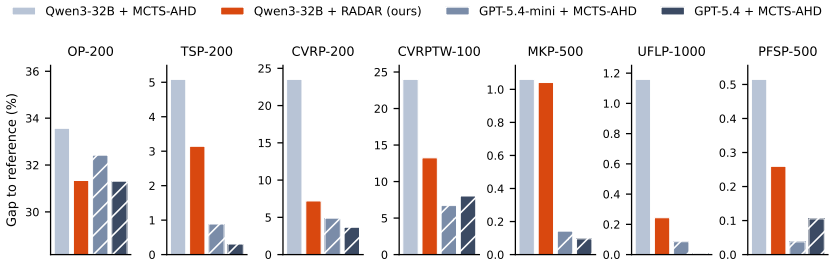



Figure 4: Cross-model gap to the best available reference (mean of four searches; lower is better). Y-axes differ by task; OP-200 starts at 28%.

### 4.4 Core-Mechanism Ablation: Racing and Diagnosis

**Study design.** We test all four combinations of the two core mechanisms on OP-200, CVRP-200, and UFLP-1000, with four searches per cell, 500 solver runs, and BAR disabled (Table 3a).

Table 3: Racing and diagnosis ablations (mean held-out scores over four searches). (a) Factorial study without BAR; subscripts give standard deviations. (b) Trace control without racing: no diagnoser (Score-only) or a diagnoser given no, swapped, or the parent's own trace. (c) MCTS-AHD and ReEvo before and after adding racing and diagnosis, with their search operators retained.

**(a)** Racing × diagnosis

| Racing | Diag. | OP-200 ↑ | CVRP-200 ↓ | UFLP-1000 ↓ |
|---|---|---|---|---|
| × | × | $56.96_{\pm 1.14}$ | $26.24_{\pm 0.70}$ | $187.71_{\pm 3.04}$ |
| ✓ | × | $58.73_{\pm 0.50}$ | $25.36_{\pm 1.09}$ | $185.47_{\pm 0.80}$ |
| × | ✓ | $58.09_{\pm 1.96}$ | $24.23_{\pm 0.77}$ | $186.32_{\pm 0.88}$ |
| ✓ | ✓ | $58.57_{\pm 0.71}$ | $23.89_{\pm 0.15}$ | $185.30_{\pm 0.39}$ |

**(b)** Trace grounding

| Condition | OP-200 ↑ | CVRP-200 ↓ | UFLP-1000 ↓ |
|---|---|---|---|
| Score-only | $56.96_{\pm 1.14}$ | $26.24_{\pm 0.70}$ | $187.71_{\pm 3.04}$ |
| No trace | $57.61_{\pm 1.16}$ | $25.14_{\pm 1.00}$ | $187.82_{\pm 1.12}$ |
| Swapped | $57.25_{\pm 0.77}$ | $25.39_{\pm 1.92}$ | $186.57_{\pm 1.52}$ |
| Trace | $\mathbf{58.09}_{\pm 1.96}$ | $\mathbf{24.23}_{\pm 0.77}$ | $\mathbf{186.32}_{\pm 0.88}$ |

**(c)** Existing methods with racing and diagnosis

| Method | R+D | OP-200 ↑ | CVRP-200 ↓ | UFLP-1000 ↓ |
|---|---|---|---|---|
| MCTS-AHD | × | 57.00 | 27.63 | 186.84 |
| | ✓ | **59.39** | **24.28** | **185.18** |
| ReEvo | × | 56.00 | 25.58 | 188.95 |
| | ✓ | **57.40** | **23.85** | **185.32** |

Racing improves the mean on all three tasks: +1.77 prize points on OP-200 and cost reductions of 0.88 on CVRP-200 and 2.24 on UFLP-1000, although the CVRP-200 gain is smaller than the spread across searches. Diagnosis also improves all three means, most on CVRP-200 (+2.01), with smaller gains on OP-200 (+1.13) and UFLP-1000 (+1.39). Combining racing and diagnosis improves the mean over the baseline without either mechanism on all three tasks. Adding BAR to the full system improves the mean on six of seven tasks; the largest gains are on CVRPTW-100 (16.04 → 15.39) and TSP-200 (11.32 → 11.16), and CVRP-200 ties (23.89 vs. 23.98; Table 12).

**Does diagnosis need the correct trace?** A diagnoser might improve edits simply by adding a second LLM call, regardless of what it reads; if the gain instead comes from resolving behavioral aliasing, it should require the parent's own trace. We compare the diagnosis-only cell (*Trace*) with three controls: *Score-only* removes the diagnoser, *No trace* withholds the report, and *Swapped* gives a real report from a behaviorally distant program on the same task (Table 3b). Across all three tasks, *Trace* has the best mean, including relative to *No trace* and *Swapped*, which also use a diagnoser and differ only in the report provided to it. This suggests that the parent's own report helps guide

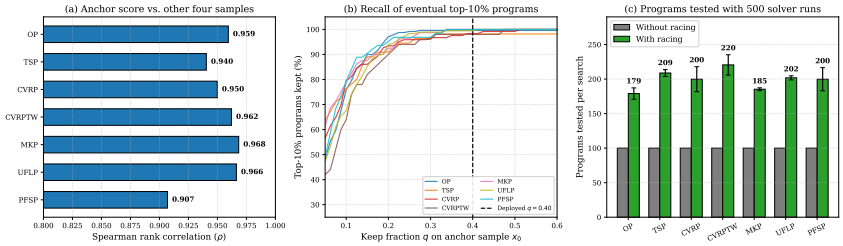



Figure 5: Racing replay on 104 fully tested searches (10,400 programs). (a) Anchor versus non-anchor score ranks (median per-search $\rho = 0.907$–$0.968$). (b) The deployed gate ($q = 0.4$) retains 99.2% of top-10% programs; stricter gates lose more. (c) Under 500 solver runs, racing tests $199 \pm 5$ programs per search versus 100 without racing.

revisions, but the differences remain descriptive with four searches per condition, especially on UFLP (186.32 vs. 186.57). Appendix F traces one such diagnosis from trace report to edit.

### 4.5 Does Racing Safely Save Solver Runs?

Because stopped programs lack full scores, we audit racing by replaying its gate on 104 searches without racing, covering 10,400 fully tested programs across seven tasks. The gate would stop 55% of them. We rank programs by their mean on the four non-anchor samples, keeping the anchor out of the outcome used to judge the gate. Figure 5 summarizes the replay (protocol in Appendix G).

The gate retains 786 of 792 top-10% programs (99.2%; 95% interval $[98.6, 99.8]$) and 103 of 104 search champions. A stopped program also removes its descendants: in 21% of searches, the champion or an ancestor would be stopped, but the best program with surviving ancestry is only 0.36% worse on average. This retrospective replay neither accounts for the extra programs funded by saved runs nor guarantees safety on other distributions; Table 3 measures the end-to-end effect.

### 4.6 Applying Racing and Diagnosis to Existing Search Methods

RADAR's two core mechanisms also apply to other search controllers. We attach the same racing gate and diagnoser to the full-program MCTS-AHD and ReEvo controllers on OP-200, CVRP-200, and UFLP-1000 (Table 3c), keeping their parent selection and search operators; the diagnosis replaces ReEvo's free-form reflection and is appended to MCTS-AHD's expansion prompt. BAR is disabled, and the budget, held-out set, and five-second limit match the main benchmark. Both controllers improve on all three tasks, with the largest relative gains on CVRP. Native and integrated runs do not share LLM completions, so these contrasts are descriptive.

## 5 Discussion and Conclusion

In this work, we introduced RADAR to address a gap in agentic solver design: final scores help identify promising programs, but reveal little about how a selected program should be improved. We argue that this ambiguity calls for using evaluation as a source of behavioral evidence as well as a ranking signal. RADAR turns trusted execution traces into specific revision instructions, while racing determines which proposals receive full evaluation and can shape later search. BAR extends this feedback loop when incremental edits stop improving a branch.

The main comparisons and controlled studies support this account. The parent's own trace produces the best mean in the trace controls, while the ablations favor racing and diagnosis under a fixed solver-run budget. Integrations with MCTS-AHD and ReEvo suggest that these mechanisms can benefit other search controllers. Across models, RADAR with an open-weight model narrows much of the gap to a stronger-model baseline on several tasks; with the commercial model, its gains are clearest where competing methods remain farther from the reference. The evidence is limited to four searches per configuration and synthetic instance generators, and RADAR uses more LLM calls than the baselines at equal solver runs.

## AI USE STATEMENT

In this work, generative AI tools were used for coding assistance, figure scaffolding, and prose editing. Literature claims, experimental design, statistical analysis, and all reported numbers were produced or verified by the authors from repository logs and frozen analysis scripts. We take responsibility for the final content.

## REPRODUCIBILITY STATEMENT

All methods use fixed random seeds, the same fixed evaluation program, and the same held-out test samples. The appendix reports every search run, the computing costs, and all settings and prompts. Code and data will be released with the supplementary material.

## REFERENCES

Mauro Birattari, Thomas Stützle, Luís Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Genetic and Evolutionary Computation Conference*, pp. 11–18, 2002.

Bin Chen, Shouliang Zhu, Beidan Liu, Yong Zhao, Tianle Pu, Huichun Li, and Zhengqiu Zhu. $A_2$DEPT: Large language model-driven automated algorithm design via evolutionary program trees. *arXiv preprint arXiv:2604.24043*, 2026a.

Minyu Chen, Song Qin, Ling-I Wu, Jianxin Xue, and Guoqiang Li. Teacher-aware evolution of heuristic programs from learned optimization policies. *arXiv preprint arXiv:2605.10634*, 2026b.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. In *Advances in Neural Information Processing Systems*, volume 37, pp. 71596–71642, 2024.

Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI)*, pp. 49–54, 1988.

Yonatan Gideoni, Sebastian Risi, and Yarin Gal. Simple baselines are competitive with code evolution. *arXiv preprint arXiv:2602.16805*, 2026.

Shuhan Guo, Nan Yin, James T. Kwok, and Quanming Yao. Nested-refinement metamorphosis: Reflective evolution for efficient optimization of networking problems. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 17398–17429, 2025.

Mirazul Haque, Petr Babkin, Farima Farmahinifarahani, and Manuela Veloso. Towards effectively leveraging execution traces for program repair with code llms. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pp. 160–179, 2025. doi: 10.18653/v1/2025.knowledgenlp-1.17.

Dong Huang et al. EffiLearner: Enhancing efficiency of generated code via self-optimization. In *Advances in Neural Information Processing Systems*, 2024a.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024b.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pp. 240–248, 2016.

Nguyen Viet Tuan Kiet, Tung Dao, Cong Dao Tran, and Huynh Thi Thanh Binh. MOTIF: Multi-strategy optimization via turn-based interactive framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37000–37008, 2026. doi: 10.1609/aaai.v40i43.41028. URL https://doi.org/10.1609/aaai.v40i43.41028.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lKEdGCoDNC.

Kelun Lei, Hailong Yang, Huaitao Zhang, Xin You, Kaige Zhang, Zhongzhi Luan, Yi Liu, and Depei Qian. PRAGMA: A profiling-reasoned multi-agent framework for automatic kernel optimization. *arXiv preprint arXiv:2511.06345*, 2025.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

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*, 2024.

Manuel López-Ibáñez and Thomas Stützle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582, 2014.

Aman Madaan, Niket Tandon, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*, 2024.

Nico Pelleriti, Sree Harsha Nelaturu, Zhanke Zhou, Zongze Li, Max Zimmer, Bo Han, and Sebastian Pokutta. What do evolutionary coding agents evolve? *arXiv preprint arXiv:2605.20086*, 2026.

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

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.

Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob Gardner, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning performance-improving code edits. In *The Twelfth International Conference on Learning Representations*, 2024.

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.

Xuan Wu, Di Wang, Chunguo Wu, Lijie Wen, Chunyan Miao, Yubin Xiao, and You Zhou. Efficient heuristics generation for solving combinatorial optimization problems using large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 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*, volume 37, 2024. doi: 10.52202/079017-1381. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

Mingxin Yu, Ruixiao Yang, and Chuchu Fan. Heuristic search as language-guided program optimization. *arXiv preprint arXiv:2602.16038*, 2026.

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 the 42nd International Conference on Machine Learning*, 2025.

Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

# A RADAR IMPLEMENTATION DETAILS

This appendix gives every setting of RADAR needed to reproduce the experiments; prompts are in Appendix B, tasks in Appendix C, and baselines in Appendix D.

## A.1 MODEL AND SAMPLING

Unless stated otherwise, searches use Qwen3-32B with hidden reasoning disabled. Every code-writing call uses temperature 1.0. Diagnoser calls that produce the three-line diagnosis (BOTTLENECK/SPEC/EVIDENCE) use temperature 1.0 with a maximum of 80 words, in all three reflection modes. All LLM calls allow up to 8192 output tokens; diagnoses are kept short by the prompt's 80-word limit. Model responses are parsed by stripping any reasoning block and extracting the first Python fenced block; a missing code block is scored as infeasible. Every search starts by evaluating a fixed seed solver (an ant-colony baseline) without charge, then rewrites it.

## A.2 BUDGETS, SPLITS, SEEDS, AND EXECUTION

In the main comparison, every method receives 500 solver runs. We also track the *program budget*, the number of programs the writer may propose (capped at 400, which is never binding in practice), and the *LLM-call budget*, the total number of writer and diagnoser calls. Each program costs one writer call, and RADAR adds at most one diagnoser call per program. Each solver run is limited to 5.0 seconds per training sample plus 1.0 second of grace for shutdown, enforced by the parent process killing the child. The search uses 5 training samples. The champion of each search is re-evaluated exactly once on a frozen set of 64 held-out test samples at 5 seconds per sample; these test scores never influence search decisions. Main comparisons use 4 independent searches per method (seeds 0–3). Each program runs in a separate forked process with working directory set to a temporary folder, single-threaded numerical libraries, and stdout/stderr discarded. The evaluator records every improving solution through a fixed reporting call, re-validates every submitted solution independently, and recomputes the objective itself, so a solver cannot inflate its own score. The prompt contract also asks for no file, network, or GPU access and a single CPU core; these requests and memory use are not enforced by an OS-level sandbox. The enforced controls are the separate process, single-threaded libraries, the parent-side kill at the time limit, and parent-side revalidation, which protect the score rather than the host. A program counts as feasible only if it returns a valid solution on every training sample.

## A.3 TREE SEARCH CONSTANTS

Programs are organized in a Monte Carlo tree. Selection uses a normalized-$Q$ UCT score with exploration constant $0.1 \times \text{eval_remain} \times \sqrt{\log(\text{parent visits} + 1)/\text{visits}}$, where $\text{eval_remain} = \max(1 - \text{progress}, 0)$ anneals exploration as the search proceeds. Value backup keeps the maximum child value; the top-3 children inform expansion; failed programs are not admitted as tree nodes. Expansion uses MCTS-AHD's operators and their prompts: e1 (a new design that differs from the given programs), e2 (a new design motivated by several programs, built on the last one as its parent), m1 (a modified version of one parent), m2 (one parent with different parameter settings), and s1 (a new design inspired by the programs along a tree path). RADAR attaches the diagnosis to e2, m1, and m2, the operators that revise one selected parent; e1 and s1 are unchanged. As a result, about 78 of every 100 programs receive a diagnosis. Progressive widening uses exponent 0.5 (a node expands while $\text{int}(\text{visits}^{0.5})$ exceeds its child count), maximum depth 10, $\epsilon = 10^{-10}$, discount 1, with running $Q$ normalization bounds initialized at 0 and $-10000$.

## A.4 RACING RULE

The anchor is fixed as training sample 0 for the whole search. The first 12 programs are fully tested on all 5 training samples to seed the anchor history $\mathcal{H}_0$; the score gate activates afterwards. Each later program runs on the anchor first: infeasible on the anchor (crash, timeout, or no valid solution) never promotes. Otherwise the score gate promotes it iff its anchor score is at or above the top-40% cutoff of all earlier anchor scores (better-first order, minimize/maximize flipped per task), i.e. keep fraction $q = 0.40$ with `WC_RACE_WARMUP=12`. A stopped program costs 1 solver run, is marked

infeasible, never enters the tree, and never becomes a parent; a promoted program costs $1 + 4$ solver runs and is scored by its five-sample mean, and the search champion is the best feasible mean.

### A.5 BRANCH-ADAPTIVE REFLECTION RULE

$$S = 0.65A + 0.35F. \tag{2}$$

Among several weightings tried in pilot runs, this split gave a stable balance between persistent stagnation and recent failures. A branch is one founder program plus all its descendants; only clean members count (feasible, tested on all 5 training samples, terminated normally). Let $k$ be the number of clean branch evaluations since the branch best was last improved. The no-gain age is $A = \min(k/12,\, 1)$: it grows by $1/12$ with each evaluation that fails to beat the branch best and saturates at 1 after 12 such evaluations, so $A$ is a scaled count, not a time. $F$ is the fraction of the four most recent clean branch evaluations that did not improve on their parent. BAR leaves the branch in the default Refine mode unless all five safeguards hold: $S \geq 0.70$; at least 8 clean branch evaluations; at least 30% of the solver-run budget consumed; at least 8 clean branch evaluations since the branch's last switch; and fewer than 3 switches on this branch so far. Switches are therefore rare (1–3 per completed search). When a switch fires, the harness chooses the mode deterministically. It chooses **Repair** if at least 2 of the last 3 edit outcomes were infeasible (recent edits are fragile), or if the search-wide best improved within the last 15 evaluations (the search is still making progress, so a conservative edit is preferred). The Repair clause (Appendix B.1) restricts the SPEC to a parameter-level change with exact values and forbids replacing any algorithmic component. Otherwise it chooses **Rethink**, whose clause requires the SPEC to replace exactly one construction, local-search, or metaheuristic component with a different method and forbids parameter tweaks. If the diagnoser returns no usable SPEC, the revision falls back to Refine and is not counted as a switch.

### A.6 EXECUTION TRACE: WHAT IS RECORDED AND SHOWN

For every training sample the evaluator records the improvement trajectory $\{(t_k, z_k)\}$ (time and score of each improving valid report), the final objective, the number of reported and valid solutions, the error string if no feasible solution, the termination mode, and counts plus the first occurrence of each rejection reason. A rendered trace report given to the diagnoser looks like this (one line per training sample):

```
inst0:  obj=52.140; first feasible 48.20@0.04s; best
52.14@3.91s; then NO improvement for 1.1s (214 valid / 231
reported tours)
inst1:  NO FEASIBLE TOUR (length_exceeded)
```

Idle time is $T_{\text{budget}} - t_K$. Rejection reasons per task: orienteering: empty tour, out-of-range node, repeated depot, duplicate node, length exceeded; vehicle routing: out-of-range customer, capacity exceeded, duplicate or missing customer; flow-shop: wrong permutation length, out-of-range or duplicate job; knapsack/location analogues for over-capacity or unassigned demand; plus malformed submission and validator-rejected fallbacks. The diagnoser additionally sees the program and score history (the 6 most recent programs plus the best program of each of up to 8 algorithm families, each shown as its one-line design description and score) and the edit history (the SPEC of each of its last 5 diagnoses paired with the measured edit outcome, so repeated failures are visible). Recording adds under 2% runtime overhead.

## B PROMPTS

Placeholders in braces are filled per parent ({problem}, {obj_desc}, {report}, {failures}, {history}, {advice}, {code}, {tb}=5 s). All prompts below are quoted exactly as issued. The diagnosis always has the three lines BOTTLENECK, SPEC, and EVIDENCE (the prompt lists SPEC before EVIDENCE). The writer receives the parent program with the task description (the MCTS-AHD operator prompt), the parent's trace report, and the SPEC line of the diagnosis, and writes a complete, executable replacement program (Appendix B.2).

### B.1 DIAGNOSER PROMPTS

Placeholders: {problem} = task name, {obj_desc} = objective description, {tb} = seconds per sample.

**Refine mode (default; used whenever BAR has not switched the branch).**

```
You are the diagnostician for an automated
solver-improvement loop on the {problem} ({obj_desc}).  Below
is the current program, the trusted measured report of
running it ({tb}s per instance), what the search has already
tried, and how your own previous advice turned out.
Measured run report (trusted harness, no model involved):
{report} {failures}{history}{advice}
Current program:  ```python {code}```
Return EXACTLY these three lines and nothing else, 80 words
total maximum:
BOTTLENECK: <the one thing most limiting this program's
objective, in one sentence>
SPEC: <the single edit to make.  Name the function or code
region and state what it should become.  Concrete enough to
implement without further decisions.>
EVIDENCE: <the specific measured number from the report,
history or advice record above that justifies it>
Decide the TYPE of edit from the measured evidence:  if
the best attempts are still improving under small changes
(see the history and your advice outcomes), SPEC ONE
parameter-level tuning edit with exact values.  If the
incumbent's level has stalled across many attempts AND the
history shows a different approach at or above its level,
SPEC ONE committed replacement.
A listed approach may be recommended again with a concrete
improvement; do not avoid an approach merely because it was
already tried.  Do not propose more than one edit.
```

**Repair and Rethink modes (same header and three lines; the harness inserts exactly one mode clause below).**

```
Return EXACTLY these three lines and nothing else, 80 words
total maximum:
BOTTLENECK: <the one thing most limiting this program's
objective, in one sentence>
SPEC: <the single edit to make.  Name the function or code
region and state what it should become.  Concrete enough to
implement without further decisions.>
EVIDENCE: <the specific measured number from the report,
history or advice record above that justifies it>
{mode_clause}
A listed approach may be recommended again with a concrete
improvement.  Do not propose more than one edit.  Do not
restate the program.
```

*Repair clause:*

```
The measured evidence says the current approach is still
paying off (or structural rewrites are currently failing
feasibility).  Your SPEC MUST be a PARAMETER-LEVEL
REFINEMENT of the current program with exact values (a
numeric constant, threshold, rate, schedule or size).  Do
NOT propose replacing any algorithmic component.
```

*Rethink clause:*

```
The measured evidence says the current approach has stalled.
Your SPEC MUST commit to ONE structural replacement (a
```

```
different construction, local-search or metaheuristic
component), chosen using the history above; prefer
an approach whose measured level is at or above the
incumbent's.  Do NOT propose a parameter tweak.
```

### B.2 WRITER PROMPT

The writer prompt starts with the unchanged MCTS-AHD prompt for the selected expansion operator (e.g. for mutation: the task description, "I have one algorithm with its code as follows," the parent program's one-line description and code, and the operator's instruction to write a new program with the required function name and interface). RADAR appends the following block, where {direction} states whether lower or higher scores are better and {spec} is the SPEC line of the diagnosis in the current reflection mode:

```
Measured run report of the algorithm above (trusted harness;
{direction}):
{report}
Diagnosis (from the measured evidence).  Implement the SPEC:
{spec}
```

## C BENCHMARK TASKS, REFERENCE RESULTS, AND CLASSICAL SOLVERS

Table 4 summarizes the seven tasks; Appendix C.1 gives each solver interface and instance generator.

**Mathematical Formulations for Exact / Optimal Solvers.**

- **UFLP-1000**: Exact MIP solved via HiGHS (`scipy.optimize.milp`) under the strong formulation: $\min_{y,x} \sum_i f_i y_i + \sum_{i,j} c_{ij} x_{ij}$ subject to $\sum_i x_{ij} = 1, x_{ij} \le y_i, y_i, x_{ij} \in \{0,1\}$. HiGHS solves all 64 test samples to **proven global integer optimality (0.00% MIP gap)** (mean 184.70).
- **MKP-500**: Exact MIP solved via HiGHS: $\max_x \sum_j p_j x_j$ subject to $\sum_j w_{ij} x_j \le C_i$, $x_j \in \{0,1\}$. HiGHS solves all 64 test samples to **proven global integer optimality (0.00% MIP gap)** (mean 105.64).
- **PFSP-500**: Taillard's analytical machine workload lower bound: $LB = \max_{1 \le m \le M} \{\min_j \sum_{k=1}^{m-1} p_{jk} + \sum_j p_{jm} + \min_j \sum_{k=m+1}^{M} p_{jk}\}$. Computed in $O(NM)$ time (mean 25832).
- **TSP-200**: Best-known tour length obtained from the minimum of OR-Tools Routing (30 s) and PyVRP/HGS (30 s) (mean 10.82).
- **CVRP-200 & CVRPTW-100**: Deep Best Known Solutions (BKS) computed via Hybrid Genetic Search (PyVRP / HGS) with 10,000 iterations and 300 s limit per test sample (CVRP: 22.37; CVRPTW: 13.59).
- **OP-200**: Provable dual upper bound computed via CP-SAT MTZ linear relaxation, warm-started with a 20 s heuristic hint (mean 85.80).

**Classical Solvers Execution.** Compute-matched classical solvers run under the same 5 s wall-clock limit per test sample:

- TSP-200: Google OR-Tools Routing (`PARALLEL_CHEAPEST_INSERTION` + `GUIDED_LOCAL_SEARCH`, 5 s; mean 11.30).
- CVRP-200: Google OR-Tools Routing (Capacity Dimension + GLS, 5 s; mean 25.44).
- CVRPTW-100: Google OR-Tools Routing (Time Windows Dimension + GLS, 5 s; mean 14.45).
- OP-200: Google OR-Tools Routing (`AddDisjunction` with node prizes, 5 s; mean 49.81).
- MKP-500: Greedy value-to-weight density ordering + add and one-for-one swap local search (5 s; mean 102.01).

- UFLP-1000: Kuehn–Hamburger greedy add followed by drop local search (5 s; mean 186.15).
- PFSP-500: NEH construction with Taillard acceleration + insertion local search (5 s; mean 25836).

Table 4: The seven benchmark problems. Every method receives 500 solver runs, with five seconds per run. The best program from each search is evaluated on 64 held-out test samples.

| Family | Task | Structural Dimension | Evaluation Metric | Splits (Train / Held-out) |
|---|---|---|---|---|
| Routing | TSP-200 | Permutation sequencing | Tour length (↓) | 5 / 64 |
| Routing | CVRP-200 | Vehicle capacity clustering | Route cost (↓) | 5 / 64 |
| Routing | CVRPTW-100 | Customer time-window deadlines | Schedule cost (↓) | 5 / 64 |
| Budgeted Routing | OP-200 | Prize vs. travel trade-off | Collected prize (↑) | 5 / 64 |
| Selection | MKP-500 | Multidimensional knapsack | Total value (↑) | 5 / 64 |
| Location | UFLP-1000 | Facility opening vs. assignment | Total cost (↓) | 5 / 64 |
| Scheduling | PFSP-500 | Flow-shop machine scheduling | Makespan (↓) | 5 / 64 |

### C.1 Solver interfaces and instance generation

All solvers are complete anytime programs: they receive instance arrays plus a wall-clock budget and a `report()` callback, and must call `report()` with each improving solution. The harness recomputes every objective and ignores infeasible or late outputs. Signatures:

- **OP-200** (maximize prize): `solve(coordinates, distance, prize, maxlen, time_budget, report)`. Tour is an integer list starting at depot 0 with unique non-depot nodes; the return leg is implied and counted against `maxlen`=5.0. 200 nodes uniform in $[0, 1]^2$; prizes $1 + \lfloor 99\, d/d_{\max} \rfloor$ normalized by depot distance $d$.
- **TSP-200** (minimize length): `solve(coordinates, distance, time_budget, report)`. 200 nodes uniform in $[0, 1]^2$, Euclidean distances, zero diagonal.
- **CVRP-200** (minimize cost): `solve(demands, coordinates, distance, capacity, time_budget, report)`. Capacity 50, demands uniform $\{1, \ldots, 9\}$, depot at $(0.5, 0.5)$.
- **CVRPTW-100** (minimize cost): `solve(demands, coordinates, distance, capacity, windows, service, time_budget, report)`. Same geometry at $n = 100$ plus deterministic time windows derived from a savings reference tour (half-widths 0.05–0.20 of tour span), service time 0.02, unit-speed Euclidean travel.
- **MKP-500** (maximize value): `solve(prize, weight, capacity, time_budget, report)`. 500 items, 5 resource dimensions, all capacities 1.0; selection is an integer list with no repeats.
- **UFLP-1000** (minimize cost): `solve(fixed_cost, coordinates, time_budget, report)`. 1000 clients, 500 facilities; fixed costs uniform $[5, 25)$ to 2 decimals; coordinates uniform in $[0, 1]^2$.
- **PFSP-500** (minimize makespan): `solve(proc_times, time_budget, report)`. Processing times $(500, 5)$ integers uniform $\{1, \ldots, 99\}$, fixed machine order 0–4.

Each task uses 5 training and 64 held-out instances from fixed seeds (training seed, validation seed, and test seed fixed per task; the UFLP split uses base seed 920260 with fixed per-split seeds).

## D Baselines and Their Full-Program Adaptation

**Shared interface.** All LLM-based methods share one interface, so that they differ only in how they search. Every method receives the same task description and the same `solve(...)` contract (Appendix C.1), including the allowed libraries and the ban on file, network, and GPU access. Every search method starts from the same seed program, an ant-colony solver, which is evaluated first. Every program is run and scored by the same trusted evaluator on the same five training samples under the same five-second limit, and code is taken from the first fenced Python block of the response.

The champion of a search is its best feasible program by mean training score, evaluated once on the 64 held-out test samples. Relative to the published fixed-skeleton versions, the adaptation therefore changes three things: (i) the task and interface description in each prompt asks for a complete solver program instead of one heuristic function inside a fixed algorithm; (ii) evaluation goes through our harness instead of the method's own evaluator; and (iii) every search is charged 100 fully tested programs (500 solver runs). Search procedures, operators, selection rules, and the remaining prompt text are kept.

**Method-specific details.**

- **MCTS-AHD**: the authors' search controller (tree, UCT selection, value backup, population, and the e1/e2/m1/m2/s1 operators with their prompts) runs unmodified; we replace only the problem description, the evaluation call, and the LLM client, and inject the seed program as the first initial node.
- **EoH**: population size 10, 20 initial programs from operator i1, then generations applying e1, e2, m1, and m2 with EoH's rank-based parent selection and objective-based deduplication. The Qwen3-32B runs use our re-implementation of these published operators, prompts, and selection rules around the shared interface; the GPT-5.4-mini runs use the EoH controller released with ReEvo's code.
- **ReEvo**: the authors' released code with population size 10, initial population 30, and mutation rate 0.5. Short-term reflection compares a worse and a better program before crossover, and long-term reflection consolidates advice before elitist mutation; the reflector is told which program is better but not their scores.
- **MOTIF**: the authors' controller, competitive MCTS, operators, contrastive reflector, and final round run unchanged. The editable surface becomes one file holding the complete `solve(...)` program (one search tree instead of one per solver component), evaluation goes through a shim to our harness, and the prompt keeps MOTIF's problem description and constraints with the `solve(...)` contract swapped in. A search is charged exactly 100 program evaluations.
- **Best-of-$N$**: $N = 100$ programs sampled independently from the task prompt with a fixed system prompt and no search feedback; the champion is the best by mean training score.

**LLM calls.** Every method spends one generation call per program. ReEvo adds reflection calls, and RADAR adds one diagnosis call for each program generated by e2, m1, or m2 (about 78 per 100 programs). Racing also lets RADAR write about twice as many programs under the same solver-run budget, so RADAR uses more LLM calls than the baselines at equal solver runs; Appendix H compares diagnosis with a score-only search at approximately equal LLM calls.

### D.1 Native fixed-skeleton versions

Table 5 compares each method in its native fixed-skeleton form (AHD) with our adapted full-program version (AAD, i.e. Automated Algorithm Design, marked †). N/A marks tasks with no native skeleton, where the published method has no fixed-skeleton formulation we could run. On the four tasks where both exist, the adaptations are consistently stronger: on CVRP-200 the native means span 29.1–35.3 cost versus 25.6–27.6 adapted (RADAR 23.98), and on OP-200 native spans 40.9–53.8 prize versus 55.6–57.5 adapted (RADAR 58.91). The AHD searches receive the same 500-solver-run budget and instance sizes, but native ACO evaluation is uncapped (8–16 s per sample versus the 5 s cap on full programs), so this comparison is context rather than a compute-matched contrast. Every AHD result in this appendix is a mean over four independent searches on the same frozen 64-sample test sets. Every independent search behind both forms is listed in Appendix M (Tables 14 and 15).

Table 5: Native fixed-skeleton versus adapted full-program mean results on the same 64 held-out test samples per problem. The Type column marks each row: AHD is the published fixed-skeleton version, AAD (marked † in Table 2) is our adaptation that emits complete programs; RADAR is reported as AAD since it writes complete programs. N/A marks tasks where the published method has no fixed-skeleton formulation. Each cell shows the mean score±standard error over four independent searches with the gap to the best available reference below it (lower gap is better). Native ACO evaluation is uncapped (8–16 s per sample versus the 5 s AAD budget), so AHD rows are context only.

| | | OP (↑) | TSP (↓) | CVRP (↓) | CVRPTW (↓) | MKP (↑) | UFLP (↓) | PFSP (↓) |
|---|---|---|---|---|---|---|---|---|
| Reference∗ | | 85.80 | 10.82 | 22.37 | 13.59 | 105.64 | 184.70 | 25832 |
| MCTS-AHD | AHD | 40.87±5.94<br>52.37% | 12.19±0.17<br>12.66% | 30.67±0.81<br>37.10% | N/A | 103.81±0.34<br>1.73% | N/A | N/A |
| | AAD | 57.00±0.30<br>33.57% | 11.37±0.06<br>5.08% | 27.63±0.32<br>23.51% | 16.85±0.94<br>23.99% | 104.52±0.33<br>1.06% | 186.84±0.52<br>1.16% | 25965±56<br>0.51% |
| EoH | AHD | 50.73±1.36<br>40.87% | 12.02±0.05<br>11.09% | 35.27±4.10<br>57.67% | N/A | 103.85±0.07<br>1.69% | N/A | N/A |
| | AAD | 55.61±1.06<br>35.19% | 11.70±0.24<br>8.13% | 26.28±0.45<br>17.48% | 19.79±0.88<br>45.62% | 104.49±0.37<br>1.09% | 185.57±0.26<br>0.47% | 25935±24<br>0.40% |
| ReEvo | AHD | 53.76±0.93<br>37.34% | 12.61±0.21<br>16.54% | 30.62±0.79<br>36.88% | N/A | 100.04±0.36<br>5.30% | N/A | N/A |
| | AAD | 56.00±0.84<br>34.73% | 11.40±0.09<br>5.36% | 25.58±0.55<br>14.35% | 18.04±0.32<br>32.74% | 104.09±0.51<br>1.47% | 188.95±2.15<br>2.30% | 26070±155<br>0.92% |
| MOTIF | AHD | N/A | 11.50±0.06<br>6.28% | 29.13±0.36<br>30.22% | N/A | N/A | N/A | N/A |
| | AAD | 57.50±0.30<br>32.98% | 11.27±0.05<br>4.16% | 27.23±0.69<br>21.73% | 19.19±0.75<br>41.21% | 103.94±0.20<br>1.61% | 188.06±0.45<br>1.82% | 26329±150<br>1.92% |
| **RADAR (Ours)** | **AAD** | **58.91±0.42<br>31.34%** | **11.16±0.05<br>3.14%** | **23.98±0.19<br>7.20%** | **15.39±0.79<br>13.25%** | **104.54±0.27<br>1.04%** | **185.15±0.09<br>0.24%** | **25899±12<br>0.26%** |

### D.2 Why we do not compare with A$_2$DEPT

A$_2$DEPT (Chen et al., 2026a) is a concurrent system synthesizing full-code heuristics using LLM-driven evolutionary search across NP-hard combinatorial domains (CFLP, CVRP, FJSP, MIS). Because A$_2$DEPT releases no code or model checkpoints and its results use the DeepSeek v3.2 backbone, which is unavailable in our compute environment, we cannot execute either system under matched conditions. For reference, A$_2$DEPT reports optimality gaps (percentage deviation from published optima, ↓) of 11.46% (CFLP), 8.79% (CVRP), 18.95% (FJSP), and 4.20% (MIS), ahead of the EoH, ReEvo, and MCTS-AHD baselines in their Table 1. Backbone, instance-library, and template differences (constructive template vs. full-code solvers) preclude a direct comparison, so we claim none and evaluate RADAR on our seven-domain benchmark instead.

## E Controlled Studies of Existing Feedback Signals

### E.1 Score-Magnitude Visibility

We use a subtractive MCTS-AHD ablation. The shown-score condition is the unmodified MCTS-AHD search. The no-score condition removes only each parent's `Objective value:` magnitude from the e1/e2/s1 prompts; parent order, ordinal lower-objective wording, MCTS $Q$/UCT selection, model, and evaluation protocol remain unchanged. A character-level prompt check confirms that shown prompts contain two numeric objective lines for e1/e2/s1 and no-score prompts contain none; m1/m2 are byte-identical.

We use Qwen3-32B (temperature 1.0, thinking off) on TSP-50, CVRP-50, MKP-100, and OP-50. Both conditions use paired seeds 0–7. Each search receives 100 evaluated programs, selects the best feasible mean-training-score champion from five training samples, and evaluates it on 64 held-out test samples (five seconds each). Table 6 and Figure 6 report the results; positive differences favor showing scores.

Table 6: Eight-pair MCTS-AHD score-visibility study. Shown retains numeric parent-score magnitudes; no-score removes only those magnitudes. $\Delta$ is direction-corrected so positive favors shown scores; intervals are paired bootstrap intervals.

| Task | Shown | No-score | $\Delta$ | 95% CI |
|---|---|---|---|---|
| TSP-50 (↓) | 5.992 | 6.005 | $+0.013$ | $[-0.037, +0.059]$ |
| CVRP-50 (↓) | 12.669 | 12.435 | $-0.235$ | $[-2.485, +2.267]$ |
| MKP-100 (↑) | 23.023 | 23.134 | $-0.111$ | $[-0.353, +0.082]$ |
| OP-50 (↑) | 14.486 | 14.581 | $-0.096$ | $[-0.659, +0.355]$ |

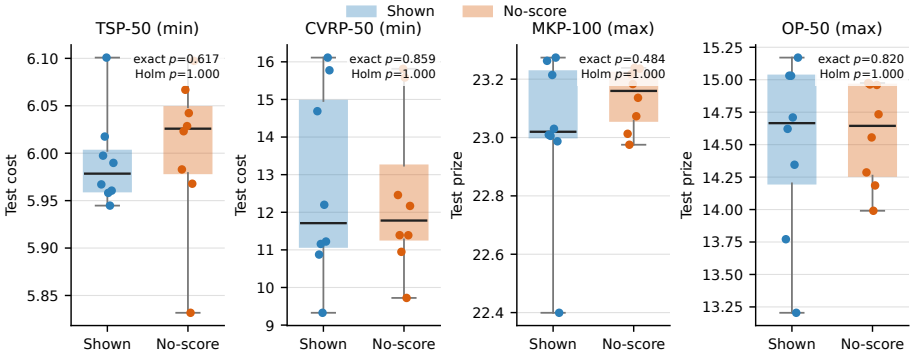



Figure 6: Eight-pair MCTS-AHD score-visibility study, shown side by side for each task. Each dot is one paired seed; boxes show the interquartile range and median; whiskers span the observed range. Insets report two-sided exact paired sign-flip $p$ values before and after Holm correction across the four tasks. Lower cost is better for TSP/CVRP, while higher prize is better for MKP/OP.

No task-level contrast survives Holm correction (all adjusted $p = 1.0$). To compare tasks with different score scales, we divide each direction-corrected mean difference by the absolute mean of the shown-score condition. This is a relative change in task score, not the optimality gap used in Table 2. The resulting changes are $+0.22\%$ for TSP, $-1.85\%$ for CVRP, $-0.48\%$ for MKP, and $-0.66\%$ for OP, where positive favors showing scores. All four are within the preregistered $\pm 5\%$ practical band. Their unweighted average is $-0.69\%$; bootstrapping the four task-level values gives a descriptive 95% interval of $[-1.51\%, -0.002\%]$. All 64 champions are feasible and use the full 100-evaluation budget. Thus, we find no stable task-level evidence that numeric parent magnitudes direct MCTS-AHD revisions once the same programs and ordering remain available. This experiment isolates score magnitude within MCTS-AHD.

### E.2 Search Specificity of ReEvo Reflection

ReEvo's short-term reflector receives two code versions labeled worse and better according to their objective values, but not the numeric values themselves, and writes advice for crossover; a long-term reflector consolidates this advice for later mutation (Ye et al., 2024). We test whether that advice is specific to its source run under four conditions on CVRP and OP (eight seeds, 64 searches total, and 106 tested programs per search): original reflection from the current run; reflection swapped from an unrelated run; the same fixed general advice in every run; and no reflection with the number of generated programs held constant. Conditions (2)–(4) reuse ReEvo's released implementation with the reflection text substituted or omitted; no new prompts are introduced.

Swapping reflection between unrelated runs causes no detectable change. Removing reflection also matches original performance while using 30% fewer LLM calls overall (34% on OP, 26% on CVRP).

Table 7: ReEvo reflection control. Cells show mean±standard error over eight independent searches (the table in Section 4.2 reports standard deviations; SE = SD/$\sqrt{8}$) on 64 held-out test samples; differences compare each condition with Original.

| Condition | CVRP Cost (↓) | Diff. vs. Orig. | $p$ | OP Prize (↑) | Diff. vs. Orig. | $p$ |
|---|---|---|---|---|---|---|
| Original | **9.887±0.14** | — | — | 14.614±0.15 | — | — |
| Swapped | 9.988±0.15 | −0.100 | .562 | **14.868±0.05** | +0.254 | .198 |
| Fixed advice | 10.618±0.11 | −0.730 | .001* | 14.659±0.13 | +0.045 | .815 |
| No reflection | 9.999±0.21 | −0.112 | .625 | 14.800±0.06 | +0.186 | .268 |

∗ Exact paired sign-flip $p = .0078$ for fixed advice on CVRP (8/8 wins for original).

Fixed general advice hurts CVRP. Thus, in this ReEvo setup, we find no detectable benefit from advice tailored to the current run. This result motivates grounding guidance in measurements.

## F Trace Grounding Control and a Worked Example

This control isolates both the presence and the content of the execution trace that feeds diagnosis. The four conditions use OP-200, CVRP-200, and UFLP-1000 with seeds 0–3, Qwen3-32B with hidden reasoning disabled, five training samples, five seconds per sample, a 400-program cap, a 500-solver-run budget, no racing, no BAR, and no post-generation refinement call. *Score-only* omits the diagnoser. *Trace* supplies the trusted per-instance improvement and validity report; *No trace* replaces it with a fixed withheld-report message; and *Swapped* supplies a same-task donor report selected to maximize behavioral distance over timing, progress, throughput, validity, rejection, termination, and instance-profile features. All other search settings are unchanged, and the three diagnosis conditions differ only in the report field. *Score-only* and *Trace* are the no-racing cells of the $2 \times 2$ study (Table 3); *No trace* and *Swapped* were run afterwards with the same budgets and search settings. Each arm reports one search per seed (seeds 0–3).

**Worked example of a trace-grounded diagnosis.** For CVRP seed 3, the trusted execution trace showed a parent that was barely feasible: on every training sample it found its first route within 0.1 s and never improved over the remaining 4.9 s, reporting only 1 valid route set out of 6–8. On training sample 0 the trace report read: `obj=28.874; first feasible 28.87@0.07s; best 28.87@0.07s; then NO improvement for 4.9s (1 valid / 6 reported route sets).` The diagnoser blamed `greedy_initial_solution()` for the poor starting points and prescribed the hybrid Clarke–Wright + local-search initialization used by an earlier program in the search history (training score 26.140). The following generated child kept the branch's aggressive-LNS + genetic-repair + annealing skeleton, swapped in exactly that initialization with a fixed 50% destruction rate, and became a new training best at 24.367 versus 26.140 ($\Delta$ −1.772; $\Delta$ −3.952 against its direct parent). This example is illustrative rather than a per-edit causal estimate; the controlled comparison is Table 3.

## G Detailed Check of the Racing Rule

**Protocol.** A stopped program is tested only on the anchor, so an audit of RADAR's own searches can compare the anchor with the full score only for programs the gate already promoted and cannot see a strong program the gate stopped by mistake. We instead replay the deployed gate on 104 searches that ran without racing, where every program was tested on all five training samples: full-program MCTS-AHD on all seven tasks (seeds 0–7) and, on OP, CVRP, and UFLP, the neither and diagnosis-only cells of Table 3 (seeds 0–7 each). The replay follows each search in generation order with the deployed rule (Appendix A.4): the first 12 programs are kept, a program whose anchor run is infeasible is stopped, and every later program is kept only if its anchor score reaches the top-40% cutoff of the earlier feasible anchor scores of that search. Programs are ranked within each search by their mean on the four non-anchor training samples, so the anchor score is never part of the label it is compared with; the top 10% are the eventual top programs and the champion is the best one.

Besides whether the gate keeps a top program itself, we check its ancestry, because a stopped program never becomes a parent. A program's ancestry survives *strictly* if it and all its ancestors are kept, and *leniently* if it is kept and at least one parent path survives (crossover programs have two parents). The *reachable best* is the best program whose ancestry survives strictly; its gap to the champion bounds what racing could lose through cut lineages in a search that is otherwise unchanged. The bound is pessimistic, since under racing the saved runs fund about twice as many programs, which this replay cannot represent.

Table 8: Racing replayed on searches where every program was fully tested. Stopped: share of programs the gate would stop. $\rho$: median per-search Spearman correlation between the anchor score and the mean of the four other training samples. Top-10% kept: programs in the top 10% of their search that the gate keeps, by itself and with strictly or leniently surviving ancestry. Champion: searches whose best program the gate keeps (by itself / strict / lenient). Gap: median and maximum percentage gap between the champion and the reachable best.

| | | | | | Top-10% kept | | | | Gap (%) | |
|---|---|---|---|---|---|---|---|---|---|---|
| Task | Searches | Programs | Stopped | $\rho$ | Itself | Strict | Lenient | Champion | Median | Max |
| OP-200 | 24 | 2,400 | 38% | 0.959 | 239/240 | 236/240 | 239/240 | 23 / 23 / 23 | 0.00 | 0.28 |
| TSP-200 | 8 | 800 | 62% | 0.940 | 54/55 | 41/55 | 48/55 | 8 / 6 / 6 | 0.00 | 1.81 |
| CVRP-200 | 24 | 2,400 | 65% | 0.950 | 120/122 | 103/122 | 112/122 | 24 / 19 / 22 | 0.00 | 13.73 |
| CVRPTW-100 | 8 | 800 | 62% | 0.962 | 49/50 | 47/50 | 48/50 | 8 / 7 / 7 | 0.00 | 6.86 |
| MKP-500 | 8 | 800 | 42% | 0.968 | 80/80 | 68/80 | 80/80 | 8 / 7 / 8 | 0.00 | 1.16 |
| UFLP-1000 | 24 | 2,400 | 63% | 0.966 | 181/182 | 113/182 | 161/182 | 24 / 15 / 21 | 0.00 | 1.90 |
| PFSP-500 | 8 | 800 | 50% | 0.907 | 63/63 | 40/63 | 60/63 | 8 / 5 / 8 | 0.00 | 0.39 |
| **Total** | **104** | **10,400** | **55%** | – | **99.2%** | **81.8%** | **94.4%** | **103 / 82 / 95** | **0.00** | **13.73** |

Table 8 gives the per-task results. The gate itself keeps 786 of the 792 top-10% programs (99.2%, 95% interval $[98.6, 99.8]$ from 2,000 resamples of whole searches) and 103 of the 104 champions (one OP search). Ancestry matters more: 81.8% of top programs ($[76.1, 86.8]$) and 82 of 104 champions keep a strictly surviving ancestry, and 94.4% and 95 of 104 keep at least one surviving parent path. The loss is small in score: the reachable best is the champion itself in 79% of searches, with a mean gap of 0.36% and a 90th percentile of 0.52%. The one large gap (13.7%) is a diagnosis-only CVRP search whose champion descends from a stopped program.

**Anchor choice.** The deployed anchor is training sample $x_0$, fixed before any search. Table 9 replays the gate with each of the five training samples in turn as the anchor, again ranking by the other four. Correlations stay between 0.87 and 0.98 except for one MKP sample (0.82), and every anchor keeps all but at most one champion per task. Top-10% retention is at least 94.5% for every anchor except one OP sample (92.9%) and the same MKP sample (85.0%), so a single unusual training sample can weaken the gate; averaging two anchor samples would reduce this risk at the cost of one more solver run per program.

Table 9: Anchor choice: each training sample in turn as the anchor, on the same 104 searches. $\rho$ range: min–max over the five anchors of the median per-search Spearman correlation with the other four samples. Recall range: min–max share of top-10% programs the gate keeps. Values in parentheses use the deployed anchor $x_0$.

| Task | $\rho$ range ($x_0$) | Top-10% recall range ($x_0$) |
|---|---|---|
| OP-200 | 0.884–0.959 (0.959) | 92.9–100% (99.6%) |
| TSP-200 | 0.916–0.940 (0.940) | 94.5–100% (98.2%) |
| CVRP-200 | 0.950–0.959 (0.950) | 95.9–100% (98.4%) |
| CVRPTW-100 | 0.957–0.967 (0.962) | 96.0–100% (98.0%) |
| MKP-500 | 0.824–0.976 (0.968) | 85.0–100% (100%) |
| UFLP-1000 | 0.961–0.970 (0.966) | 98.4–100% (99.5%) |
| PFSP-500 | 0.876–0.907 (0.907) | 96.8–100% (100%) |

**Keep-fraction sweep.** Table 10 varies the keep fraction $q$ with the deployed anchor. Keeping 10% loses 20–36% of the top programs and keeping 20% still loses up to 10% on several tasks. At $q = 0.30$ every task keeps at least 96%, and at $q = 0.40$ at least 98%, losing at most two programs

per task. Keeping half recovers the remaining CVRP, CVRPTW, and UFLP misses but not the OP and TSP ones, while spending more of the budget on full evaluations.

Table 10: Keep-fraction sweep: percentage of top-10% programs the gate keeps when keeping a fraction $q$ of programs (anchor $x_0$, same 104 searches). The deployed rule uses $q = 0.40$.

| $q$ | OP | TSP | CVRP | CVRPTW | MKP | UFLP | PFSP |
|---|---|---|---|---|---|---|---|
| 0.10 | 75.8 | 76.4 | 79.5 | 64.0 | 78.8 | 67.6 | 79.4 |
| 0.20 | 97.1 | 90.9 | 93.4 | 90.0 | 92.5 | 94.0 | 95.2 |
| 0.30 | 99.6 | 96.4 | 96.7 | 96.0 | 98.8 | 98.9 | 96.8 |
| **0.40** | **99.6** | **98.2** | **98.4** | **98.0** | **100.0** | **99.5** | **100.0** |
| 0.50 | 99.6 | 98.2 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |

**Programs from RADAR's own searches.** On the 2,026 programs that RADAR without BAR tested on all five samples (warm-up and promoted programs only), the anchor correlates with the five-sample mean at $\rho = 0.871$–$0.954$ across tasks. These correlations are lower than above, likely because the gate has already removed most weak programs, which narrows the range of scores; this pool cannot measure how many strong programs the gate stops, which is why the replay above is the main check.

**What racing stops: failure modes pruned after one run.** Stopped programs cost one anchor run instead of five and never enter the tree as parents, so the saved runs fund new programs while weak directions attract no further edits. The stopped set spans three recurring modes. (1) Outright failures: programs that crash, time out, or return no valid solution on the anchor. (2) Constraint-violating programs: programs whose anchor outputs are rejected, such as the inspected OP program that omits the depot return (average score 47.36) or deadline-violating CVRPTW insertions, including runs that fail completely. Such programs fall far below the top-40% anchor cutoff. (3) Feasible but uncompetitive directions: clean runs whose anchor score is valid yet below the cutoff, including families that keep polishing a weak core without improving. The deployed racing rule does not attempt to distinguish these modes; for promoted programs, diagnosis can use execution traces together with scores and history to make the revision feedback more specific.

## H Program Efficiency under Comparable LLM Inference

This experiment asks whether diagnosis still helps when the baseline receives the extra LLM calls as additional programs. The RADAR arm is the diagnosis-only configuration of Table 3: racing and BAR are off, so each of its 100 programs is tested on all five training samples within the 500 solver runs. The score-only arm is native full-program MCTS-AHD, given about 181 programs so that its LLM calls match RADAR's; since each program is fully tested, this also gives score-only about 905 solver runs instead of 500. The comparison is therefore matched in LLM calls and favors the baseline in solver runs; it uses CVRP-200, OP-200, and TSP-200 with eight paired seeds.

Table 11: Comparison under similar total LLM use, with eight paired searches per task. Score-only tests ≈181 programs using ≈181 LLM calls, while RADAR tests 100 programs using ≈178 calls (≈78 diagnoses plus 100 writer calls). Positive values favor RADAR; per-task $p$-values are two-sided exact paired sign-flip tests.

| Task | Mean Gain | Median Gain | Paired Wins | $p$-value |
|---|---|---|---|---|
| CVRP-200 | $+0.375$ | $+1.834$ | 5/8 | .664 |
| OP-200 | $+0.925$ | $+1.534$ | 5/8 | .336 |
| TSP-200 | $+0.166$ | $+0.142$ | 7/8 | .070 |
| **Pooled (24 paired runs): 17/24 wins, one-sided sign test $p = .032$ ($p = .064$ two-sided)** Standardized effect $+0.414$, 95% CI $[+0.018, +0.809]$ | | | | |

Despite testing 45% fewer programs, RADAR has the better average on all three tasks (Fig. 7). Across the 24 paired runs, it wins 17 times. The combined one-sided test gives $p = .032$ ($p = .064$ for a two-sided test), with a standardized effect of $+0.414$ and a 95% uncertainty interval of $[+0.018, +0.809]$.

The eight-run results for each individual task remain uncertain, but all three average differences favor RADAR. This suggests that better-guided edits can be more useful than simply generating more programs.

Measured LLM calls on the 2×2 cells (OP/CVRP/UFLP, 12 runs per condition, 500 solver runs each): score-only ≈100 calls, diagnosis-only ≈178 calls (≈78 diagnoses plus 100 writer calls), racing-only ≈211 calls. Token counts were not logged.

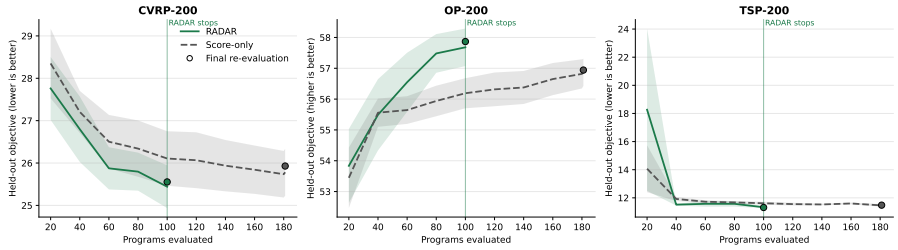



Figure 7: Program efficiency under matched LLM calls. RADAR stops at 100 evaluated programs, while the score-only search continues to 181. Lines show held-out checkpoints during the search, with bands indicating ±1 SEM over 8 paired seeds; circles mark the final re-evaluated scores reported in Table 11.

**LLM-call accounting.** Call counts are logged per search in the reevaluation records. Under the comparable-inference experiment (Appendix H), RADAR's 100 evaluated programs required 170–184 LLM calls per search (task means 180.5/175.9/178.5: 100 writer calls plus 70–84 diagnoses), while the score-only baseline required 181–183 calls (task means 181.0/182.0/181.1): ≈178 vs. ≈181 calls, so LLM compute is approximately comparable rather than exactly matched. Per-call token counts were not logged; diagnoses are short (bounded under 80 words).

## I Branch-Adaptive Reflection: Switch Evidence and Ablation

We call each time BAR moves a branch out of the default Refine mode a *switch*; it issues exactly one Repair or Rethink diagnosis. As a first example, consider TSP-200 seed 1 of the BAR runs (Fig. 8). Its training best (node P16, 11.2406, set at evaluation 16) stayed unbeaten through evaluation 160. At evaluation 160 the founder branch P1 switched to Rethink with $S = 1.000$ ($A = 1.0$, $F = 1.0$): 31 clean branch evaluations, budget progress 0.648, 8 clean evaluations since its earlier Repair switch at evaluation 46, and 1 prior switch, passing every safeguard in Appendix A.5. The training best then moved to 11.1901 by evaluation 180 (node P177) and 11.1253 by evaluation 200 (node P194); the run's final held-out score is 11.0633. The search used 2 switches in total, within the reported 1–3 per search.

**Second example: CVRPTW-100 seed 3.** The same frozen batch gives a Rethink case on a different task (Fig. 9). Branch P1's best (node P58, 16.974, set at evaluation 58) saw no improvement for 8 clean evaluations and no new best in the last 15, with $S = 0.783$ ($A = 8/12 \approx 0.67$, $F = 1.0$): 31 clean branch evaluations, budget progress 0.55, 0 prior switches, passing every safeguard. Because the search as a whole had not improved in the last 15 evaluations, BAR chose Rethink at evaluation 99 on parent P68 (train 17.92). The Rethink child itself was rejected, but the branch best fell to 14.128 (node P113) by evaluation 113, about a 17% step within 14 evaluations of the trigger. The run finished with held-out 14.52, the best of its task.

**BAR ablation on all seven tasks.** Table 12 is the BAR ablation: RADAR without BAR versus the current RADAR with BAR (four seeds each, same frozen config: Qwen3-32B, temperature 1.0, 500 solver runs, racing top-40% after 12 warm-up programs, 400-program cap). BAR has the better average on 6 of 7 tasks, with CVRP about tied.

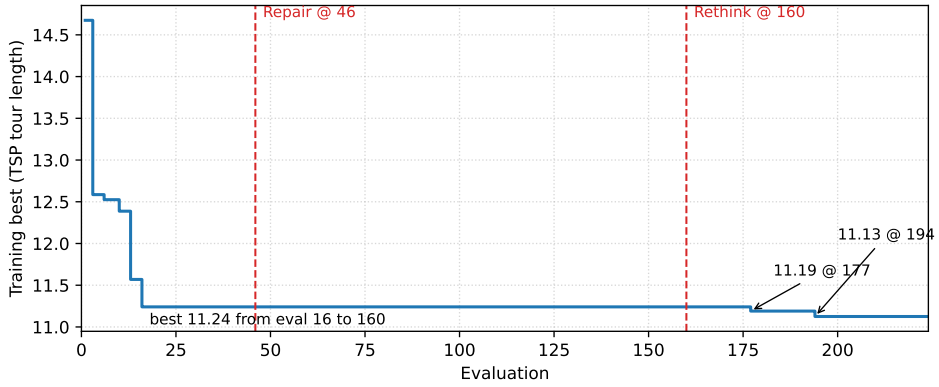



Figure 8: Training best over evaluations for TSP-200 seed 1. Dashed lines mark the two BAR switches (Repair at 46, Rethink at 160). The best sat at 11.24 from evaluation 16 to 160, then moved to 11.19 and 11.13 after the Rethink switch.

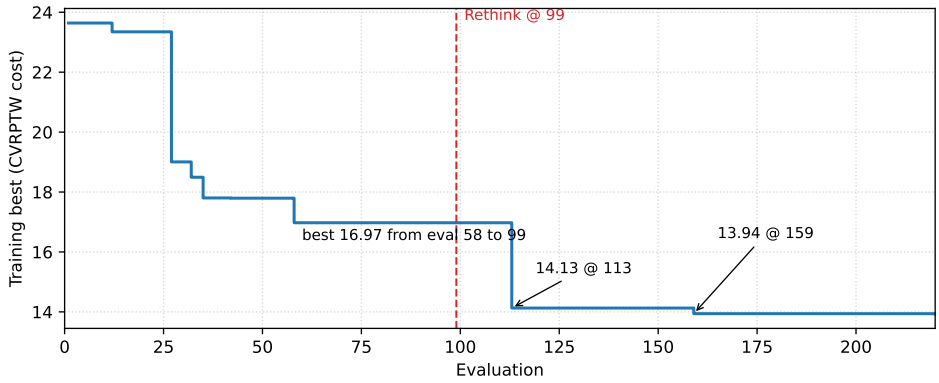



Figure 9: Training best over evaluations for CVRPTW-100 seed 3. The dashed line marks the Rethink switch at 99. The best sat at 16.97 from evaluation 58 to 99, then fell to 14.13 at 113 and 13.94 at 159.

Table 12: BAR ablation: held-out means with versus without BAR (four seeds each, same frozen config: Qwen3-32B, temperature 1.0, 500 solver runs, racing top-40% after 12 warm-up programs, 400-program cap). Gain is direction-corrected so positive favors With BAR.

| Task | Without BAR | With BAR | Gain |
|---|---|---|---|
| OP-200 (↑) | 58.57 | 58.91 | +0.34 |
| TSP-200 (↓) | 11.32 | 11.16 | +0.16 |
| CVRP-200 (↓) | 23.89 | 23.98 | −0.09 |
| CVRPTW-100 (↓) | 16.04 | 15.39 | +0.65 |
| MKP-500 (↑) | 104.26 | 104.54 | +0.28 |
| UFLP-1000 (↓) | 185.30 | 185.15 | +0.15 |
| PFSP-500 (↓) | 25922 | 25899 | +23.2 |

## J Testing Changes to the Problem Definition

We test whether RADAR still works when the orienteering problem (OP) is presented in an unfamiliar form. We use three versions:

1. `op`: the standard problem and standard variable names.
2. `opdisg`: the same problem with unfamiliar variable names.
3. `opsplice`: a modified problem in which repeated visits to one of 12 spatial groups reduce the available prize by a factor of $\gamma = 0.3$.

Across 72 runs with Qwen3-32B on 100-node instances (OP-100; 100 programs per search and 8 searches for each method–problem pair), the final results for Best-of-$N$, score-only search, and RADAR were:

- On `op`: 27.50 (Best-of-$N$) / 29.55 (Score-only) / 31.12 (RADAR).
- On `opdisg`: 28.06 (Best-of-$N$) / 29.41 (Score-only) / 31.98 (RADAR).
- On `opsplice`: 8.50 (Best-of-$N$) / 8.80 (Score-only) / 9.20 (RADAR).

Changing only the variable names produces no significant difference ($p > .18$). On the modified problem, all eight final RADAR programs are valid and follow the new prize-discount rule. This suggests that RADAR is not limited to copying one standard solver pattern. It does not show that the LLM has no relevant prior knowledge.

## K Additional Results with GPT-4.1-mini

Table 13: Mean held-out test scores with GPT-4.1-mini on three tasks, averaged over seeds 0–2. Arrows indicate the preferred direction; bold marks the best mean per task.

| Task | MCTS-AHD | RADAR | ReEvo | EoH |
|---|---|---|---|---|
| CVRP-200 (↓) | 23.868825 | **23.234294** | 23.295601 | 23.440012 |
| OP-200 (↑) | 55.482627 | **57.670136** | 56.823883 | 54.791306 |
| UFLP-1000 (↓) | 184.855294 | **184.738224** | 184.860625 | 184.839371 |

## L Closing the Gap to Proprietary Models

Figure 4 combines the Qwen3-32B and GPT-5.4-mini results from Table 2 with an additional GPT-5.4 run of full-program MCTS-AHD. The additional run uses the same protocol: four searches per task, 500 solver runs, five training samples, five seconds per sample, and the same 64 held-out test samples. The gap-closure percentage compares the reduction from Qwen3-32B MCTS-AHD to Qwen3-32B RADAR with the difference between Qwen3-32B and GPT-5.4 using MCTS-AHD. It is 67% on CVRPTW-100 in addition to the tasks reported in the main text. On OP-200, Qwen3-32B RADAR has a 31.34% gap, compared with 32.42% for GPT-5.4-mini MCTS-AHD and 31.31% for GPT-5.4 MCTS-AHD. Gap-closure percentages are sensitive to small mean differences when the two MCTS-AHD models are already close.

## M PER-SEED BENCHMARK RESULTS

Table 14 reports the result of every independent search on the fixed set of 64 held-out test samples.

Table 14: Result of every independent search on the 64 held-out test samples. The final column gives the mean±standard error. Every cell uses seeds 0–3. Bold marks the best and underline the second-best mean per task.

| Task | Method | Seed 0 | Seed 1 | Seed 2 | Seed 3 | Mean±SEM |
|---|---|---|---|---|---|---|
| OP-200 (↑) | MCTS-AHD | 57.47 | 56.47 | 56.48 | 57.56 | 57.00±0.30 |
| | EoH | 53.88 | 55.08 | 58.69 | 54.79 | 55.61±1.06 |
| | ReEvo | 54.98 | 55.74 | 54.83 | 58.43 | 56.00±0.84 |
| | Best-of-$N$ | 45.38 | 46.90 | 46.90 | 48.32 | 46.88±0.60 |
| | MOTIF | 58.20 | 56.79 | 57.74 | 57.28 | <u>57.50±0.30</u> |
| | **RADAR (Ours)** | 59.32 | 59.38 | 57.66 | 59.29 | **58.91±0.42** |
| TSP-200 (↓) | MCTS-AHD | 11.26 | 11.52 | 11.41 | 11.30 | 11.37±0.06 |
| | EoH | 11.25 | 11.81 | 11.40 | 12.33 | 11.70±0.24 |
| | ReEvo | 11.54 | 11.29 | 11.57 | 11.21 | 11.40±0.09 |
| | Best-of-$N$ | 11.35 | 11.39 | 11.39 | 11.39 | 11.38±0.01 |
| | MOTIF | 11.20 | 11.34 | 11.19 | 11.37 | <u>11.27±0.05</u> |
| | **RADAR (Ours)** | 11.12 | 11.06 | 11.29 | 11.18 | **11.16±0.05** |
| CVRP-200 (↓) | MCTS-AHD | 27.88 | 27.51 | 28.32 | 26.82 | 27.63±0.32 |
| | EoH | 27.05 | 26.26 | 26.78 | 25.02 | 26.28±0.45 |
| | ReEvo | 24.07 | 25.72 | 26.69 | 25.86 | 25.58±0.55 |
| | Best-of-$N$ | 24.87 | 25.15 | 23.86 | 26.12 | <u>25.00±0.46</u> |
| | MOTIF | 28.36 | 27.43 | 25.24 | 27.89 | 27.23±0.69 |
| | **RADAR (Ours)** | 24.27 | 24.34 | 23.53 | 23.78 | **23.98±0.19** |
| CVRPTW-100 (↓) | MCTS-AHD | 17.14 | 15.89 | 15.00 | 19.36 | <u>16.85±0.94</u> |
| | EoH | 18.85 | 19.31 | 22.39 | 18.60 | 19.79±0.88 |
| | ReEvo | 18.58 | 17.13 | 18.21 | 18.26 | 18.04±0.32 |
| | Best-of-$N$ | 15.90 | 19.79 | 15.00 | 16.86 | 16.89±1.04 |
| | MOTIF | 17.20 | 20.22 | 20.44 | 18.90 | 19.19±0.75 |
| | **RADAR (Ours)** | 14.67 | 17.77 | 14.62 | 14.52 | **15.39±0.79** |
| MKP-500 (↑) | MCTS-AHD | 104.06 | 105.50 | 104.26 | 104.27 | <u>104.52±0.33</u> |
| | EoH | 105.59 | 104.32 | 103.96 | 104.11 | 104.49±0.37 |
| | ReEvo | 104.53 | 104.88 | 104.37 | 102.58 | 104.09±0.51 |
| | Best-of-$N$ | 103.29 | 103.76 | 103.53 | 104.39 | 103.74±0.24 |
| | MOTIF | 104.05 | 103.36 | 104.08 | 104.26 | 103.94±0.20 |
| | **RADAR (Ours)** | 105.34 | 104.35 | 104.25 | 104.21 | **104.54±0.27** |
| UFLP-1000 (↓) | MCTS-AHD | 185.36 | 187.40 | 187.73 | 186.86 | 186.84±0.52 |
| | EoH | 186.34 | 185.40 | 185.20 | 185.34 | <u>185.57±0.26</u> |
| | ReEvo | 185.65 | 191.50 | 193.67 | 184.96 | 188.95±2.15 |
| | Best-of-$N$ | 185.82 | 185.37 | 189.12 | 188.50 | 187.20±0.94 |
| | MOTIF | 186.94 | 187.88 | 189.11 | 188.30 | 188.06±0.45 |
| | **RADAR (Ours)** | 185.05 | 185.41 | 185.10 | 185.04 | **185.15±0.09** |
| PFSP-500 (↓) | MCTS-AHD | 25927 | 25872 | 26127 | 25934 | 25965±56 |
| | EoH | 25917 | 25913 | 26006 | 25906 | <u>25935±24</u> |
| | ReEvo | 26535 | 25921 | 25913 | 25911 | 26070±155 |
| | Best-of-$N$ | 26125 | 26025 | 26665 | 26279 | 26273±140 |
| | MOTIF | 26022 | 26325 | 26232 | 26735 | 26329±150 |
| | **RADAR (Ours)** | 25890 | 25929 | 25872 | 25905 | **25899±12** |

Table 15: Native fixed-skeleton AHD: every independent search on the same frozen 64-sample sets as the AAD rows. Native ACO runs take 8–16 s per sample versus the 5 s AAD budget.

| Task | Method | Seed 0 | Seed 1 | Seed 2 | Seed 3 | Mean±SEM |
|---|---|---|---|---|---|---|
| OP-200 (↑) | MCTS-AHD | 40.15 | 25.64 | 43.23 | 54.48 | 40.87±5.94 |
| | EoH | 48.75 | 48.52 | 54.35 | 51.31 | 50.73±1.36 |
| | ReEvo | 51.52 | 55.96 | 53.27 | 54.30 | 53.76±0.93 |
| TSP-200 (↓) | MCTS-AHD | 12.61 | 11.90 | 11.95 | 12.31 | 12.19±0.17 |
| | EoH | 11.95 | 11.93 | 12.10 | 12.10 | 12.02±0.05 |
| | ReEvo | 13.17 | 12.67 | 12.24 | 12.36 | 12.61±0.21 |
| | MOTIF | 11.66 | 11.40 | 11.44 | 11.49 | 11.50±0.06 |
| CVRP-200 (↓) | MCTS-AHD | 31.29 | 31.82 | 28.27 | 31.29 | 30.67±0.81 |
| | EoH | 31.33 | 29.54 | 47.39 | 32.81 | 35.27±4.10 |
| | ReEvo | 29.37 | 30.78 | 29.54 | 32.80 | 30.62±0.79 |
| | MOTIF | 28.67 | 30.16 | 29.14 | 28.56 | 29.13±0.36 |
| MKP-500 (↑) | MCTS-AHD | 104.28 | 104.03 | 102.82 | 104.12 | 103.81±0.34 |
| | EoH | 104.00 | 103.93 | 103.71 | 103.76 | 103.85±0.07 |
| | ReEvo | 101.05 | 100.01 | 99.42 | 99.68 | 100.04±0.36 |

## N Why Gains Differ Across Tasks

The results also suggest why gains differ across problems. On TSP, PFSP, and MKP, programs written from the initial prompt are already within about 1–5% of the reference result, leaving little room to improve. On CVRPTW and OP, some initial programs fail to return to the depot or violate deadlines. RADAR records these failures and can ask for an edit that addresses them. This explanation is based on inspection after the experiments and is not a separate cause-and-effect test.

We inspect programs generated directly from the initial prompt to understand why some tasks leave little room for improvement:

- **TSP-200, PFSP-500, and MKP-500**: The LLM directly writes well-known methods: nearest-neighbor with 2-opt and Or-opt for TSP, NEH for PFSP, and value-to-weight ordering for MKP. Best-of-$N$ is therefore already within $5.2\%$, $1.7\%$, and $1.8\%$ of the respective reference results (Table 2a).
- **OP-200**: One inspected program checks whether adding a node stays within the tour-length limit but forgets to include the distance back to the depot. Its average score is $47.36$, compared with $58.91$ for RADAR.
- **CVRPTW-100**: Simple distance-based insertion can violate arrival deadlines. Best-of-$N$ varies widely across searches (15.00–19.79, Table 14). RADAR records the deadline rejections and reaches an average cost of $15.39$ with valid final programs in every run.

## O Statistical Procedures

Each main-table cell averages 4 independent searches; the±range is the standard error (standard deviation divided by $\sqrt{4}$), i.e. how much that average would vary across independent sets of 4 searches. Flow-shop values are rounded to integers because decimals are noise at that scale.

**Small-scale controls.** The score-visibility study uses 8 paired searches per task: an exact two-sided sign-flip test over all $2^8 = 256$ sign assignments, a paired bootstrap interval from 100000 resamples (fixed seed 20260912), and a Holm correction across the four tasks. The ReEvo reflection study uses 8 paired searches per condition and reports paired two-sided $t$-tests against Original; the exact sign-flip test is given where noted.

**Ablations.** Factorial: all 4 racing × diagnosis combinations on OP/CVRP/UFLP, 4 searches per cell at 500 runs; each cell reports the raw mean with the standard deviation of the four searches in that cell. Trace control: four arms on the same three tasks, 4 searches each, racing and BAR disabled. Cost-matched: 8 paired searches per task on CVRP/OP/TSP, exact paired sign-flip tests per task and a one-sided sign test pooled over the 24 pairs (Appendix H).