# FROM AGENT HARNESSES TO COP EXPERTS: AUTOMATED SKILL EVOLUTION FOR CROSS-PROBLEM COMBINATORIAL OPTIMIZATION

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

## ABSTRACT

General-purpose agents built on agent loops and execution harnesses are becoming the default interface for complex tasks, yet they remain generalists. In combinatorial optimization (CO), existing agentic systems typically rely on manually engineered workflows tailored to each problem, which limits reuse and scalability. This gap raises a central question: can a general agent automatically acquire skills that turn it into an expert solver across heterogeneous COPs? We propose an automated skill-evolution method that learns reusable behavioral guidance from solver-development experience, allowing the agent to discover and refine its own skills from indirect feedback rather than relying on handcrafted workflows. Our key result is that a skill evolved solely from TSP development evidence transfers to CVRP, JSSP, and 3D-BPP. Across these three unseen COPs, it substantially outperforms no skill, the initial skill, and state-of-the-art skill-evolution methods such as SkillOpt, which optimizes agent skills as a trainable external state through validation-gated text-space edits. These findings suggest a promising route to general agents that can automatically become experts across diverse combinatorial optimization problems.

## 1 INTRODUCTION

General-purpose agents are becoming increasingly adept at complex tasks. They can reason about goals, employ tools, execute code, and refine their actions in response to feedback (Yao et al., 2023; Shinn et al., 2023; Wang et al., 2024; Yang et al., 2024). In LLM-based combinatorial optimization (COP), this growing competence is expanding the role of agents from merely generating heuristics or algorithmic components (Romera-Paredes et al., 2024; Liu et al., 2024; Ye et al., 2024; Novikov et al., 2025) to constructing and iteratively refining complete, executable solvers (Chen et al., 2026b; Sun et al., 2026). This latter task requires agents to integrate problem analysis and mechanism design with implementation, evaluation, and revision. General capability alone, however, does not automatically make an agent a COP expert. Existing agentic COP systems compensate for this gap by equipping agents with domain-specific expertise, for example by assigning specialized roles, organizing work into staged workflows, or providing interfaces to dedicated solvers (Xiao et al., 2024; Ahmaditeshnizi et al., 2024; Zhang et al., 2026b). Although these systems demonstrate the potential of agents for optimization, much of their specialization remains tied to the system design, and it may require new engineering for each additional COP. We therefore need a way for a general-purpose agent to acquire reusable COP expertise without heavy, problem-specific design.

Skills offer a lighter path to such specialization. A manually written skill, however, does not guarantee the behavior its author intended. Because agents interpret natural-language guidance instead of executing it as exact code, even guidance that seems clear to its author can lead to unintended behavior (Webson & Pavlick, 2022; Zhan et al., 2024). A solving skill should therefore be evaluated and revised against the behavior and outcomes it actually produces. Recent skill evolution methods pursue this idea by distilling trajectories, converting diagnoses into update directions, or optimizing skills through validation-gated edits (Ni et al., 2026; Wang et al., 2026; Yang et al., 2026c). In COP-related work, OptSkills learns reusable workflows for optimization modeling and solving, with correctness as its primary objective (Yang et al., 2026a), while SkillOpt can generate and validate text revisions (Yang et al., 2026c) but does not specify which solver-development behaviors those revisions should target, since solver quality does not directly reveal how the skill should change.



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.

These studies demonstrate that external guidance can be refined through experience, but they leave open how to revise such guidance for complete COP solver development.

We propose an automated *skill evolution* method for solving combinatorial optimization problems (COPs). The core idea is to convert indirect result feedback into reusable behavioral guidance, enabling an agent to build complete code-based solvers and continually improve how it builds them. The framework assigns three distinct roles. First, a *solver agent* develops a complete solver under the current *solving skill*. Second, a fixed *skill evolver* analyzes the resulting evidence and proposes a revision to that *solving skill*. Third, an external performance rule selects between the current and candidate skills. Throughout this process, only the *solving skill* evolves, while the *solver agent*, the *skill evolver*, and the selection rule remain fixed. The main challenge is that feedback is indirect. The *solving skill* guides how the agent develops a solver, but external evaluation scores only the resulting solver. To bridge this gap, the *skill evolver* traces backward from an observed performance issue to the relevant solver code and the corresponding development decision, and then examines how the current *solving skill* shaped that decision. The resulting revision is expressed as general solver-development guidance, and it is retained only if it improves the solver in a fresh development run.

Using TSP as the source COP for skill evolution, we find that the evolved skill substantially outperforms no skill, the initial skill, and SkillOpt, a state-of-the-art skill-evolution baseline. Remarkably, the same frozen skill, without further evolution, also substantially outperforms these alternatives on three heterogeneous COPs, namely CVRP, JSSP, and 3D-BPP, and continues to produce competitive solvers under additional language models and agent harnesses. This cross-problem transfer suggests that heterogeneous COPs may share a common core logic in solver design, which the evolved skill captures as reusable behavioral guidance. These results demonstrate that solver-development experience can be distilled into guidance that is transferable to different problems and executors.

## 2 Related Work

**Automated Solver Development.** Approaches that evolve programs directly use executable performance to optimize programs or heuristic components (Romera-Paredes et al., 2024; Liu et al., 2024; Ye et al., 2024; Novikov et al., 2025). Recent methods also synthesize full algorithms beyond fixed heuristic templates (Chen et al., 2026a; Yazdani et al., 2026). EvoPH extends this line by co-evolving the prompts that guide generation and the heuristic population they produce (Liu et al., 2026b). Although these methods increasingly adapt the search process, they still optimize the resulting program, so the expertise they acquire remains tied to that program and does not transfer as reusable development knowledge. Agentic COP research broadens the unit of automation to the development process, where specialized agents, staged workflows, and search procedures coordinate formulation, implementation, debugging, and improvement (Xiao et al., 2024; Ahmaditeshnizi et al., 2024; Li et al., 2025; Zhang et al., 2026b). HeuriGym and CO-Bench evaluate this broader capability through executable agent-produced heuristics and algorithms (Chen et al., 2026b; Sun et al., 2026). This progress automates more of solver development, yet it leaves expertise embedded in either the optimization program or the engineered development system. In both cases, expertise remains bound to a specific program or a hand-designed pipeline, so adapting it to a new COP typically requires new engineering.

**Skill Evolution.** Skill Evolution, by contrast, renders experience reusable as editable external guidance for an agent. Existing methods differ in how execution evidence becomes guidance. Distillation-based methods consolidate lessons across trajectories into transferable skills or knowledge bases (Ni et al., 2026; Tang et al., 2026). Diagnosis-driven methods use failures or losses to identify deficiencies and propose targeted edits, as in EvoSkill, SkillGrad, and SkillForge (Alzubi et al., 2026; Wang et al., 2026; Liu et al., 2026a). Validation-driven methods generate or select skill revisions using held-out or verifier feedback, as in SkillOpt, CoEvoSkills, DIVE, and the verifier-guided Trace2Skill for EDA agents (Yang et al., 2026c; Zhang et al., 2026a; Xiong et al., 2026; Du & Pinckney, 2026). In all three categories, skill updates are tied directly to the execution or validation evidence of the skill-using agent, since the same agent both follows the skill and produces the evaluated output. Our setting introduces a further separation. The skill guides an agent that develops a solver, while external quality is observed only on the resulting solver. The revision evidence therefore does not reveal how the skill itself should change. Existing methods cannot be applied directly. Our backward trace

bridges this indirect evidence path by connecting solver quality to the development decisions the skill was meant to shape.

**Experience and Skill Learning for Combinatorial Optimization.** Recent COP work has begun to externalize modeling experience. AlphaOPT extracts solver-verified insights from failed formulations and refines their applicability conditions (Kong et al., 2025), while OptSkills clusters structurally related problems and distills successful trajectories into reusable modeling-and-solving workflows (Yang et al., 2026a). OptiSkill stores solver-verified modeling experience in an evolving skill bank (Zhao et al., 2026). These modeling methods target whether individual natural-language optimization problems are correctly formulated and solved, and measure success primarily by formulation or solving accuracy. Related heuristic-design systems extract evolution strategies, accumulate mechanism-level knowledge, or explicitly search over reusable design hypotheses (Yang et al., 2025; 2026b; Nguyen et al., 2026). Our setting differs in both its object and its signal. A skill guides a long development process that produces a separately evaluated executable solver, so solver quality does not directly reveal which guidance should change.

## 3 METHOD

Our objective is to develop a reusable *solving skill* that transforms a general-purpose agent into an expert COP solver, which is expected to be transferable, generalizing to unseen and heterogeneous COPs as well as to alternative agent-loop harnesses. To this end, we introduce a skill evolution method that refines the *solving skill* through evaluation feedback and solver-development evidence. Within this framework, only the *solving skill* evolves. A fresh solver agent develops each solver, a fixed *evolver skill* guides evidence analysis and candidate revision, and an external performance rule selects between current and candidate skills. Each evolution round proceeds in three stages. First, a complete solver is developed under the current *solving skill*. Second, evidence from that solver is used to propose a revision to the skill. Third, the candidate skill is evaluated through fresh solver development, and the better version is selected to enter the next round. The following subsections detail the solver agent's development process, how performance and process evidence guide skill revision, and how fresh solver performance determines skill selection.

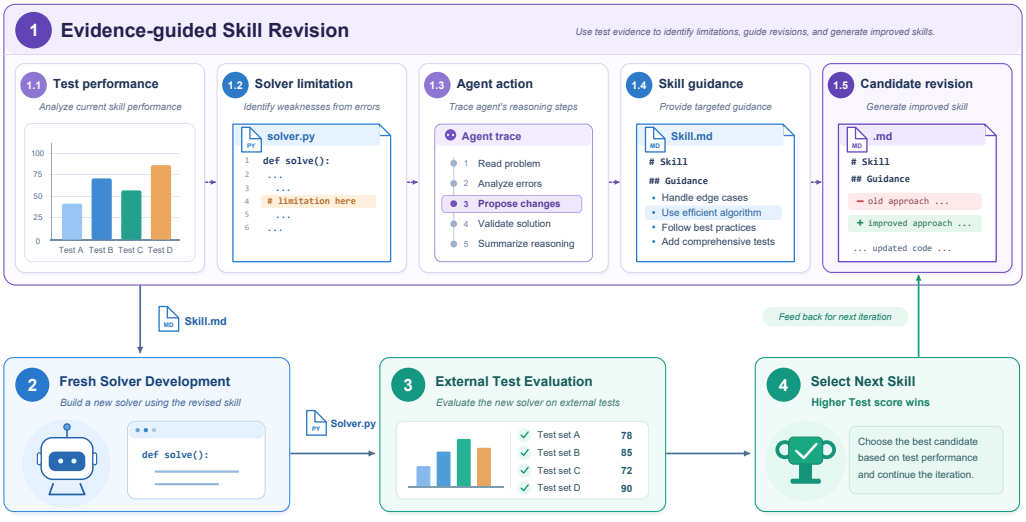



Figure 1: Overview of the Skill Evolution workflow, including evidence-guided revision, fresh solver development, external evaluation, and skill selection.

### 3.1 SOLVER DEVELOPMENT UNDER THE SOLVING SKILL

We manually wrote the initial skill $S_0$ to make iterative solver development an explicit requirement, rather than an unstated practice that the agent might or might not adopt. The skill specifies a ReAct-

style loop (Yao et al., 2023): the agent implements a solver, runs it on training instances, examines both its performance and search behavior, and revises the code before testing again. This process is illustrated in Figure 2. The loop remains unchanged across skill versions; as the skill evolves, we refine only the guidance for interpreting results and choosing revisions within the loop, not the loop itself. To assess the agent's independent problem-solving ability, we prohibit web search and external solvers and restrict implementation to Python's standard library, NumPy, Numba, and basic SciPy numerical functions.

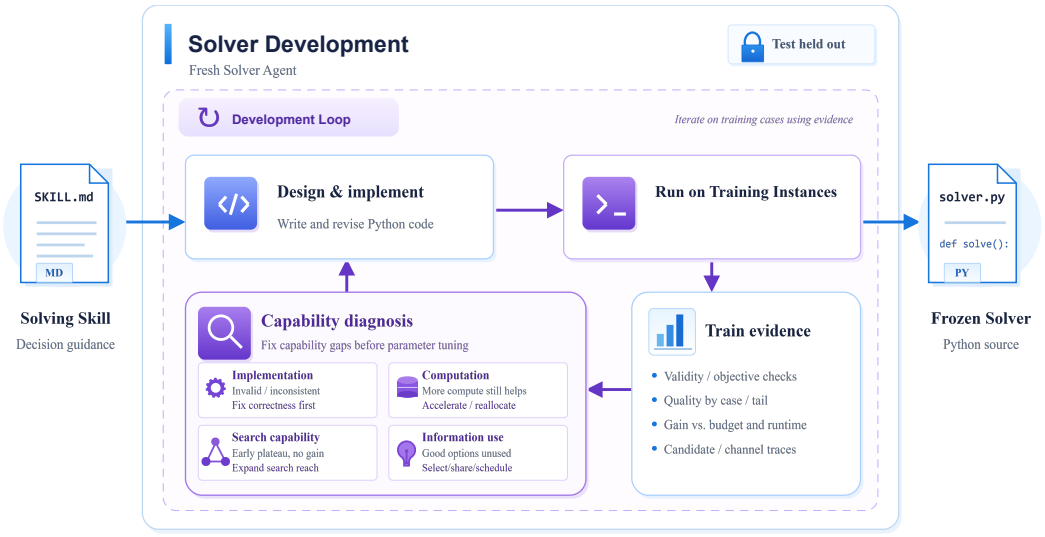



Figure 2: Solver development under the Solving Skill, with the Test set held out.

### 3.2 From Evaluation Feedback to Skill Revision

After a solver agent develops a solver under the current Solving Skill, an external evaluator runs it on evaluation instances. The Skill Evolver receives the resulting feedback and solver code together with the agent's development records. Feedback shows where the solver is weak, but not which skill guidance to revise. It therefore relates a weak outcome to the solver design, the agent's decision behind it, and the relevant guidance in the current skill. If that guidance appears insufficient, it drafts a problem-general revision. This trace motivates an edit without establishing that the Solving Skill alone caused the weakness.

**Identify the performance shortfall.** The Skill Evolver examines evaluation feedback by instance and by problem features to locate where the completed solver performs poorly. The shortfall may be concentrated at a particular scale or structure, appear in a tail of high-gap instances, or persist across rounds. The mean gap is used to select skill versions, while this observed shortfall gives the subsequent trace a specific outcome to explain.

**Trace the design and decision.** The Skill Evolver examines the final solver code for a design limitation consistent with the evaluation pattern. The code shows which mechanisms were implemented, while the runs show how the solver behaved within its development budget. The Skill Evolver then uses the development trajectory to locate when the agent introduced or retained that design. It checks what observations were available at that point and how the agent explained its choice. This reconstructs an observable development decision without claiming access to the agent's unrecorded reasoning.

**Check the Solving Skill.** The Skill Evolver compares this development decision with the corresponding guidance in the current Solving Skill $S_t$ (for the first iteration, it will check the initial Solving Skill $S_0$). Guidance that is missing, ambiguous, conflicting, or too general to act on may provide a target for revision. Clear guidance that the agent did not follow, an isolated coding error, or

an incomparable run does not by itself justify changing the skill. When the available evidence does not support a skill-level explanation, the Skill Evolver is instructed to leave that issue untouched.

**Revise the Solving Skill.** When the corresponding guidance in the current Solving Skill $S_t$ appears insufficient, the Skill Evolver revises it to address the development decision identified in the trace. The edit may clarify an instruction, resolve conflicting advice, or make a missing diagnostic consideration explicit. It is framed in terms of problem structure, training observations, and resource limits, not benchmark identities, evaluation outcomes, or the source solver's algorithm. For example, a low-throughput solver that continues to improve on training instances may motivate clearer guidance on checking computational efficiency before replacing its search mechanism. The source solver informs the revision but is not copied into the skill. The edit produces a candidate skill $C_t$ for evaluation.

### 3.3 Skill Selection by Fresh Solver Performance

A fresh solver agent develops a new solver under the candidate skill $C_t$ using the same COP task, training instances, and development budget as the current version. The external evaluator measures the new solver's mean gap on the evaluation instances. We compare that gap with the recorded gap for the retained skill $S_t$ and adopt $C_t$ only if the gap decreases. The agent's or Skill Evolver's explanation does not affect this decision.

Let $\mathrm{Score}(S)$ be the negative mean evaluation gap obtained when skill $S$ was evaluated. A higher score therefore means a lower mean gap. The update rule is

$$S_{t+1} = \begin{cases} C_t, & \mathrm{Score}(C_t) > \mathrm{Score}(S_t), \\ S_t, & \mathrm{Score}(C_t) \leq \mathrm{Score}(S_t). \end{cases} \tag{1}$$

A tie or an increase in mean gap leaves $S_t$ unchanged.

## 4 Experiments and Results

The experiments aim to address three questions: 1)**Evolution on the source COP.** Does evolution improve the Solving Skill so that an agent develops a better solver on the COP used for evolution? 2)**Transfer to unseen COPs.** Does the Evolved Skill also improve solver quality on unseen, structurally heterogeneous COPs? 3)**Deployment across models and agent harnesses.** How well does the frozen Solving Skill retain its performance when installed in different models and agent harnesses?

### 4.1 Experimental Setup

**Dataset.**

| COP | Train (synthetic) | Test (public) |
|---|---|---|
| TSP | 42; 6 sizes, 7 spatial structures | 77 TSPLIB EUC_2D |
| CVRP | 50; X-generator and structured cases | 100 CVRPLIB-X |
| JSSP | 48; 3 instance families | 80 Taillard/ 80 DMU |
| 3D-BPP | 64; 8 MPV classes | 320 MPV320 |

**Access and controls.** Solver agents are given access to the task definition, training instances, and references, but never to testing instances. During evolution, only the Skill Evolver receives feedback from the TSP test set. All other test sets are reserved exclusively for final evaluation. In our main experiments, we use GPT-5.6-Sol with Codex. Each fresh run is offline and isolated and has no access to solvers (e.g., LKH). All runs use matched task data, dependencies, and development budgets (Appendices C.1 and B).

**External evaluation.** An external evaluator verifies feasibility and recomputes objectives under task- and size-specific time limits ranging from 10 to 180 seconds per instance. For TSP, CVRP, and JSSP, we report mean gaps relative to frozen reference solutions. For 3D-BPP, we report the number of

bins used. Final evaluations are based on three fresh solver-development runs, whereas evolution candidates use a single run (Appendices C.2 and C.3).

### 4.2 TSP Skill Evolution

Our method and SkillOpt both start from the same Initial Skill and run for ten rounds with GPT-5.6-Sol and Codex. In each round, one candidate skill is evaluated through fresh solver development, and the next round begins from the version retained by that method's selection gate as designed in Section 3.3. Figure 3 plots the gaps of all candidate skills, including those that were rejected. The plotted lines therefore do not trace the retained skills. Starting from a shared baseline of 1.69%, the selected versions ultimately reach 0.44% (ours, round 8) and 1.01% (SkillOpt, round 9). Appendix Tables 7 and 8 report the revisions and selection decisions.

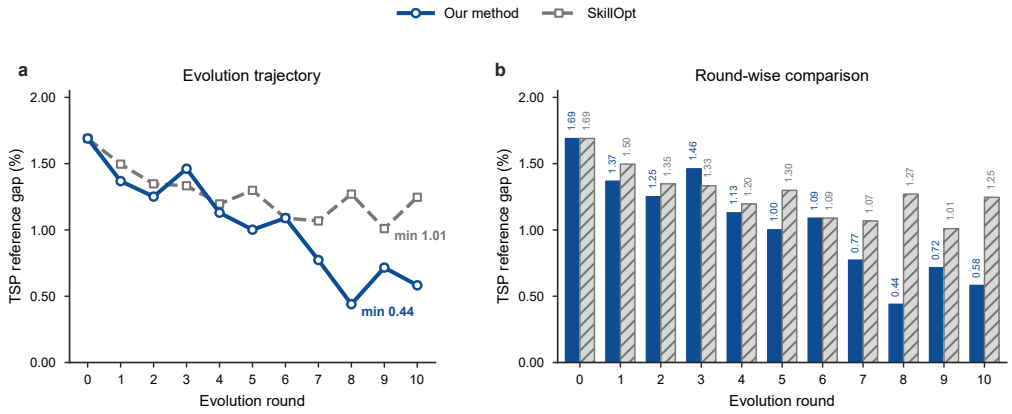



Figure 3: TSP Test mean gap of each round's candidate Solver (lower is better).

### 4.3 Comparison of Evolution Processes and Evolved Skills

**Revision targeting.** Both methods rely on execution feedback and reject candidate skills, but they differ in how they identify a skill edit. Our Skill Evolver traces a weakness observed during evaluation through the developed solver design and the agent's recorded decisions back to the relevant guidance in the Solving Skill before editing. In contrast, SkillOpt aggregates rollout successes and failures into bounded edits and applies a validation gate, without requiring this backward trace (Yang et al., 2026c). The retained TSP gaps were close at round 6 (ours: 1.002%; SkillOpt: 1.090%) but diverged by round 8 (0.440% versus 1.068%). During this interval, our accepted edits targeted channel allocation and distinct search moves, whereas SkillOpt's round-7 edit addressed threshold stability and its round-8 deletion of full-train validation was rejected (Appendix Tables 7 and 8).

Table 1: Evidence requested for selected Solver-development decisions by the three Skill texts.

| Development decision | Initial Skill | SkillOpt Skill | Our evolved Skill |
|---|---|---|---|
| **Target selection** | Features and weak groups | Recurring weak regions | Feature–capability link; counterexamples |
| **Stagnation diagnosis** | Qualitative compute/search check | Curves and neighborhood success | Paired-budget throughput/reach probes |
| **Mechanism validation** | Changed moves; component tests | Target weak region; matched on/off test | Distinct moves; staged validation |
| **Channel allocation** | Time, candidates, and gains | Recent gain/time; reserve exploration | Short- and long-run return; protect novel channels |

**Retained skill guidance.** Table 1 compares the evidence required to make four solver-development decisions. The full skill texts are linked in Appendix E.1. The Initial Skill already addresses these decisions, and both evolved versions make its guidance more operational. Compared with SkillOpt, our method makes two checks more explicit: paired probes comparing throughput with search reach, and evidence that a new mechanism produces distinct moves. This comparison concerns only the written guidance, not agent compliance or the effects of individual clauses. Section 4.4 evaluates the complete skills.

Table 2: Solver quality across model–Agent configurations using Evolved Skill, with Initial Skill under the main configuration as a reference. Lower is better.

| Skill | Model–Agent configuration | TSP gap (%) | CVRP gap (%) | JSSP gap (%) | 3D-BPP bins |
|---|---|---|---|---|---|
| Initial Skill | GPT 5.6-Sol + Codex | 1.69 | 4.49 | 10.68 | 9985.3 |
| Evolved Skill | GPT 5.6-Sol + Codex | 0.52 | 1.37 | 3.48 | 9786.3 |
| Evolved Skill | GPT 5.6-Sol + Pi Agent | 0.54 | 1.76 | 4.56 | 9787.7 |
| Evolved Skill | DeepSeek V4-Flash + Codex | 0.63 | 2.86 | 5.68 | 9909.7 |

### 4.4 Final Skill Evaluation Across COPs

Across all four COPs, the Evolved Skill attains the best average among the four skill conditions (Table 3), with both evolved skills frozen before evaluation. Against SkillOpt, it reduces the mean gap by **47%** on TSP (0.52% vs. 0.98%), **62%** on CVRP (1.37% vs. 3.63%), and **50%** on JSSP (3.48% vs. 6.92%), and it uses **89.4 fewer bins** on 3D-BPP (9786.3 vs. 9875.7). Against the Initial Skill, the reductions are larger still: 69% on TSP (1.69% $\rightarrow$ 0.52%), 69% on CVRP (4.49% $\rightarrow$ 1.37%), 67% on JSSP (10.68% $\rightarrow$ 3.48%), and **199.0 fewer bins** on 3D-BPP. The Evolved Skill also leads across every one of the 32 reported size, scale, shape, and class groups, and on several JSSP groups it matches the best-known upper bound, reaching **0.00%** on Taillard $100\times20$ and 0.03% on Taillard $50\times15$.

The four skill conditions are No Skill, Initial Skill, SkillOpt, and Evolved Skill, and both evolved skills were frozen before evaluation. TSP supplied evolution and version-selection feedback, whereas CVRP, JSSP, and 3D-BPP were unseen, so no target-task feedback or TSP solver artifacts entered their solver-development runs.

TSP gaps use proven optima, whereas CVRP and open JSSP cases use frozen best-known upper bounds, and closed JSSP cases use optima. The three gap averages weight Test instances equally rather than the displayed size or family groups. For 3D-BPP we report total bins across all 320 instances. Figure 3 shows one evolution run per candidate, whereas Table 3 averages three fresh development runs, so its TSP result need not equal the selected run's 0.44%.

Published methods provide benchmark context rather than matched-run evidence (Ye et al., 2024; Chen et al., 2026c; Zhang et al., 2026b; Zago et al., 2026; Corsini et al., 2024; Mahvash et al., 2018; Gonçalves & Resende, 2013; Feng et al., 2015). Against LLM-based TSP and CVRP methods, solvers developed with the Evolved Skill reduce the mean gap by roughly an order of magnitude, reaching 0.52% versus 13.10% (ReEvo) and 12.24% (DRAGON) on TSP, and 1.37% versus 15.11% (ReEvo) and 4.95% (AFL) on CVRP. Against learning-based JSSP methods, it reaches 3.48%, compared with 12.59% (NDS-S) and 10.45% (SLIM $\mathrm{GM}_{512}$).

The 3D-BPP comparison is especially informative because HGA-NR and BRKGA-BPP are expert-designed solvers specialized for packing, whereas our Evolved Skill is a single frozen general-purpose skill that was evolved only on TSP and transferred without any target-task feedback. Despite this asymmetry, the Evolved Skill stays close to both. It beats HGA-NR on three of the eight MPV classes (Classes 2, 6, and 8), ties it on one (Class 3), and trails it on the remaining four, finishing within **17.3 bins** of HGA-NR on the mean total. It ties BRKGA-BPP on two classes (Classes 4 and 5) and trails on the other six, but every loss is at most 0.08 bins, so the mean totals differ by only **9.3 bins**. A skill evolved from a single source COP thus already generalizes strongly enough to approach expert-designed, problem-specific solvers on an unseen packing task. Appendix C.1 details coverage.

Table 3: Solver quality on public benchmarks. Lower values are better. The best result in each row is shown in bold, and the second best is underlined.

**TSP — TSPLIB EUC_2D** (mean gap %; node scale)

| **Benchmark** | **Scale** | No Skill | Initial | SkillOpt | **Ours** | ReEvo | DRAGON |
|---|---|---|---|---|---|---|---|
| TSPLIB | 50–500 | 2.14 | 0.07 | 0.25 | **0.02** | 8.17 | 9.74 |
| | 501–1,000 | 3.30 | 0.96 | 0.73 | **0.36** | 16.61 | 11.24 |
| | 1,001–2,000 | 4.19 | 1.74 | 1.29 | **0.63** | 18.23 | 15.24 |
| | 2,001–5,000 | 5.05 | 2.72 | 2.34 | **1.52** | 21.20 | 19.37 |
| | 5,001–20,000 | 11.48 | 10.89 | 3.50 | **2.43** | 18.62 | 15.42 |
| *Average* | | 3.75 | 1.69 | 0.98 | **0.52** | 13.10 | 12.24 |

**CVRP — CVRPLIB-X** (mean gap %; customer scale)

| **Benchmark** | **Scale** | No Skill | Initial | SkillOpt | **Ours** | ReEvo | AFL |
|---|---|---|---|---|---|---|---|
| X | 100–200 | 7.58 | 3.59 | 2.62 | **0.52** | 8.77 | 1.93 |
| | 201–500 | 6.41 | 4.79 | 3.83 | **1.33** | 14.88 | 5.20 |
| | 501–1,000 | 6.45 | 4.68 | 4.04 | **2.00** | 19.81 | 6.66 |
| *Average* | | 6.68 | 4.49 | 3.63 | **1.37** | 15.11 | 4.95 |

**JSSP — Taillard and DMU** (mean gap to best-known UB %; jobs $\times$ machines)

| **Family** | **Shape** | No Skill | Initial | SkillOpt | **Ours** | NDS-S | SLIM ($GM_{512}$) |
|---|---|---|---|---|---|---|---|
| Taillard | $15\times15$ | 18.84 | 3.09 | 2.75 | **0.84** | 3.20 | 6.50 |
| | $20\times15$ | 22.97 | 5.56 | 4.78 | **2.22** | 5.50 | 8.80 |
| | $20\times20$ | 20.54 | 5.14 | 4.29 | **1.86** | 5.20 | 9.00 |
| | $30\times15$ | 23.16 | 6.32 | 4.53 | **2.20** | 7.40 | 10.60 |
| | $30\times20$ | 28.07 | 9.10 | 7.45 | **3.91** | 10.80 | 12.70 |
| | $50\times15$ | 17.47 | 6.48 | 0.48 | **0.03** | 8.10 | 4.90 |
| | $50\times20$ | 19.88 | 8.06 | 2.60 | **0.53** | 10.20 | 7.60 |
| | $100\times20$ | 9.82 | 5.13 | 0.12 | **0.00** | 9.70 | 2.10 |
| DMU | $20\times15$ | 27.95 | 8.69 | 7.01 | **3.63** | 8.60 | 11.30 |
| | $20\times20$ | 27.19 | 8.46 | 7.18 | **3.47** | 9.20 | 12.30 |
| | $30\times15$ | 30.61 | 15.36 | 8.37 | **4.76** | 17.10 | 14.00 |
| | $30\times20$ | 31.98 | 15.17 | 8.44 | **7.09** | 17.20 | 15.80 |
| | $40\times15$ | 26.76 | 15.38 | 10.28 | **4.70** | 18.80 | 10.90 |
| | $40\times20$ | 33.36 | 18.67 | 13.26 | **6.90** | 20.50 | 14.80 |
| | $50\times15$ | 24.96 | 18.63 | 12.84 | **5.45** | 24.90 | 10.60 |
| | $50\times20$ | 31.25 | 21.59 | 16.38 | **8.03** | 25.10 | 15.00 |
| *Average* | | 24.67 | 10.68 | 6.92 | **3.48** | 12.59 | 10.45 |

**3D-BPP — MPV320** (mean bins by class; summary row is mean total bins $\downarrow$)

| **Benchmark** | **Class** | No Skill | Initial | SkillOpt | **Ours** | BRKGA-BPP | HGA-NR |
|---|---|---|---|---|---|---|---|
| MPV | 1 | 33.00 | 32.70 | 32.12 | 31.88 | 31.80 | **31.58** |
| | 2 | 32.65 | 32.23 | 31.78 | 31.39 | **31.35** | 31.43 |
| | 3 | 32.95 | 32.54 | 32.01 | 31.65 | **31.63** | 31.65 |
| | 4 | 73.53 | 73.50 | 73.61 | 73.50 | 73.50 | **73.33** |
| | 5 | 18.20 | 18.17 | 17.87 | 17.60 | 17.60 | **17.43** |
| | 6 | 24.23 | 24.17 | 24.03 | 23.76 | **23.73** | 23.90 |
| | 7 | 15.63 | 15.24 | 14.87 | 14.59 | **14.58** | **14.58** |
| | 8 | 21.38 | 21.09 | 20.63 | 20.29 | **20.25** | 20.35 |
| *Mean total bins* | | 10062.0 | 9985.3 | 9875.7 | 9786.3 | 9777 | **9769** |

### 4.5 Evolved Skill Generalized to Other Harness and Model Combinations

We take the skill evolved entirely from GPT 5.6-Sol with Codex and deploy it, without any further evolution, under two alternative configurations: GPT 5.6-Sol with Pi Agent and DeepSeek V4-Flash with Codex. The Solving Skill, tasks, data, tools, and evaluation protocol remain fixed across all runs. This setup tests whether the evolved skill captures reusable COP expertise rather than a recipe tailored to one specific model or harness.

Table 2 reports these results alongside the Initial Skill under the main GPT 5.6-Sol–Codex configuration as a reference. Under the main configuration, the Evolved Skill reaches 0.52% on TSP, 1.37% on CVRP, 3.48% on JSSP, and 9786.3 bins on 3D-BPP. Switching the harness to Pi Agent with GPT 5.6-Sol preserved this performance almost exactly, changing the gaps by only 0.02 to 1.08 percentage points and the 3D-BPP total by 1.3 bins. Switching the model to DeepSeek V4-Flash with Codex degraded performance more noticeably, by 0.11 to 2.20 points and 123.3 bins, yet the resulting solver still improves over the Initial Skill by 1.06 points on TSP, 1.63 on CVRP, 5.00 on JSSP, and 75.6 bins on 3D-BPP. Because neither alternative configuration has its own paired control, this comparison does not estimate the relative skill gain per executor. It nevertheless shows that a skill evolved from one model–harness pair transfers to others and continues to turn a general-purpose agent into a strong COP solver. Detailed group-level results appear in Appendix A.

## 5 Discussion

**Cross-Problem Generalization.** Evolved solely from TSP, the frozen Solving Skill improved fresh Solver development in routing, scheduling, and packing problems. Its advantage over SkillOpt held across independent runs and all reported strata, without transferring the TSP Solver or using target-task feedback. This supports reusable development guidance rather than a specific recipe.

**Evolution Dynamics.** Both our method and SkillOpt rejected poorer candidates, so rollback alone cannot explain our larger late-round gain. Our backward trace connects a performance weakness to Solver design, a recorded Agent action, and the relevant Skill guidance before revision. The resulting checks for distinct search reach, delayed returns, and staged validation are consistent with better revision targeting (Section 4.3). Each candidate was evaluated once, and individual rules were not ablated.

**Cross-Executor Deployment.** Prior paired evaluations find that Skill gains vary across model–harness configurations (Li et al., 2026). Here, both alternative configurations using the frozen Evolved Skill outperformed Initial Skill under the main configuration on every reported metric. But it must be acknowledged that there is slight fluctuation in performance when switching harness, and fallback occurred while switching model.

**Scope and Future Directions.**

The clearest observed behavioral change was a better stopping boundary, where agents without a Skill often stopped at the first feasible Solver and the Initial Skill could still leave a persistent gap, whereas the Evolved Skill prompted further mechanisms and controlled tests before stopping, alongside stronger search coverage, comparisons, rollback, and structural validation. Because our method still guarantees neither correct nor efficient individual decisions, and because stopping remains a cost–quality tradeoff in which spending 50% more tokens to reduce the gap from 5% to 1% may or may not be worthwhile, future work should test whether decisions follow evidence, consider alternatives, and produce informative experiments while balancing Solver quality against development cost. A further open question concerns the training signal itself, since evolution depends on Train-set coverage and difficulty, which we did not vary, and narrow or easy Train sets can hide Solver deficiencies, while three runs do not remove generation variance. Addressing these issues will require paired executor controls, additional runs, controlled Train sets, and broader source and target COPs. Our evidence therefore supports cross-problem guidance within the tested setting, not universal COP expertise.

## 6 Conclusion

This paper introduced an automated Skill Evolution method for developing complete COP Solvers with a general Agent. The method turns an issue observed in evaluation results into problem-general guidance by tracing it backward through the relevant Solver code, the Agent action that produced that code, and the corresponding guidance in the current Skill. We evolved the Skill using only TSP development evidence and then froze it for evaluation on CVRP, JSSP, and 3D-BPP. Across these three heterogeneous COPs, Evolved Skill outperformed No Skill, Initial Skill, and SkillOpt, and the same frozen Skill remained usable with additional model–Agent configurations. These results show that Solver-development experience can be distilled into reusable behavioral guidance without transferring the source Solver or handcrafting a workflow for each target problem. Automated Skill Evolution therefore offers a lightweight route for specializing general Agents as COP solvers.

### Reproducibility statement

Section 3 defines the learning object and describes evidence diagnosis, Skill revision, and version selection. Section 4.1 specifies the comparison conditions and execution constraints. The appendix reproduces the final Evolved Skill, Skill Evolver, execution templates, and the four task contracts; it also reports the ten-round evolution audit and documents data splits and evaluation metrics.

### Ethics statement

This study evaluates language-model Agents on synthetic and public combinatorial-optimization instances; it does not analyze human-participant or personal data. Generated code may consume substantial computational resources or behave unsafely. We therefore disable network and external-solver access and execute all programs in a controlled environment. All authors will read and comply with the ICLR Code of Ethics before submission.

### AI use statement

In this work, generative AI Agents are both the object of study and a component of the method. They are used to analyze COPs, generate and revise Solvers, and evolve the external Skill; these uses are described in the method and experimental setup.

## References

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 2024.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

Bin Chen, Shouliang Zhu, Beidan Liu, Yong Zhao, Tianle Pu, Huichun Li, and Zhengqiu Zhu. A2DEPT: Large language model-driven automated algorithm design via evolutionary program trees, 2026a. URL https://arxiv.org/abs/2604.24043.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, Samitha Samaranayake, Carla P. Gomes, and Zhiru Zhang. HeuriGym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, 2026b.

Shengkai Chen, Zhiguang Cao, Jianan Zhou, Yaoxin Wu, Senthilnath Jayavelu, Zhuoyi Lin, Xiaoli Li, and Shili Xiang. DRAGON: LLM-driven decomposition and reconstruction agents for large-scale combinatorial optimization. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems*, pp. 2809–2817, 2026c. doi: 10.65109/SBAY6258.

Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. In *Advances in Neural Information Processing Systems*, volume 37, pp. 105528–105551. Curran Associates, Inc., 2024. doi: 10.52202/079017-3349.

Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998. doi: 10.1016/S0377-2217(97)00019-2.

Zijian Du and Nathaniel Pinckney. Trace2Skill: Verifier-guided skill evolution for long-context eda agents, 2026. URL https://arxiv.org/abs/2605.21810.

Xuehao Feng, Ilkyeong Moon, and Jeongho Shin. Hybrid genetic algorithms for the three-dimensional multiple container packing problem. *Flexible Services and Manufacturing Journal*, 27(2–3):451–477, 2015. doi: 10.1007/s10696-013-9181-8.

José Fernando Gonçalves and Mauricio G. C. Resende. A biased random-key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500–510, 2013. doi: 10.1016/j.ijpe.2013.04.019.

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, Shenhao Wang, Haris Koutsopoulos, Hai Wang, Cathy Wu, and Jinhua Zhao. AlphaOPT: Formulating optimization programs with self-improving LLM experience library, 2025. URL https://arxiv.org/abs/2510.18428.

Dong Li, Xujiang Zhao, Linlin Yu, Yanchi Liu, Wei Cheng, Zhengzhang Chen, Zhong Chen, Feng Chen, Chen Zhao, and Haifeng Chen. SolverLLM: Leveraging test-time scaling for optimization problem via LLM-guided search. In *Advances in Neural Information Processing Systems*, volume 38, pp. 110867–110897, 2025. doi: 10.52202/085713-3343.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

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*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. SkillForge: Forging domain-specific, self-evolving agent skills in cloud technical support. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4763–4768. Association for Computing Machinery, 2026a. doi: 10.1145/3805712.3808466.

Yihong Liu, Junyi Li, Hongyu Lu, Xin Zhao, and Ji-Rong Wen. Experience-driven reflective co-evolution of prompts and heuristics for autonomous algorithm design. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 4981–4995. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.findings-acl.245.

Batoul Mahvash, Anjali Awasthi, and Satyaveer Chauhan. A column generation-based heuristic for the three-dimensional bin packing problem with rotation. *Journal of the Operational Research Society*, 69(1):78–90, 2018. doi: 10.1057/s41274-017-0186-7.

Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000. doi: 10.1287/opre.48.2.256.12386.

Viet Tuan Kiet Nguyen, Dinh Pham Bui, Van Tung Dao, Cong Dao Tran, and Thi Thanh Binh Huynh. Back to the beginning of heuristic design: Bridging code and knowledge with LLMs, 2026. URL https://arxiv.org/abs/2605.06123.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

Alexander Novikov, Ngan Vu, 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, 2025. URL https://arxiv.org/abs/2506.13131.

Gerhard Reinelt. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4): 376–384, 1991. doi: 10.1287/ijoc.3.4.376.

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*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377.

Weiwei Sun, Shengyu Feng, Shanda Li, and Yiming Yang. CO-Bench: Benchmarking language model agents in algorithm search for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(39):33126–33134, 2026. doi: 10.1609/aaai.v40i39.40596.

Éric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. doi: 10.1016/0377-2217(93)90182-M.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. WikiSkill: Compiling agent experience into persistent knowledge for skill evolution, 2026. URL https://arxiv.org/abs/2608.27454.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017. doi: 10.1016/j.ejor.2016.08.012.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. SkillGrad: Optimizing agent skills like gradient descent, 2026. URL https://arxiv.org/abs/2605.27760.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 2024.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2300–2344. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.naacl-main.167.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations*, 2024.

Siheng Xiong, Ali Payani, Oguzhan Gungordu, and Faramarz Fekri. DIVE: Unlocking self-improvement in frozen language models through diversity-driven skill evolution, 2026. URL https://arxiv.org/abs/2608.12486.

Haochen Yang, Ke Zhao, Mengyuan Ma, Xingyu Lu, Xiangfeng Wang, and Hong Qian. OptSkills: Learning generalizable optimization skills from problem archetypes via cluster-based distillation, 2026a. URL https://arxiv.org/abs/2605.29829. Accepted to Findings of EMNLP 2026.

Jiancong Yang, Xinan Chen, Rong Qu, Zishang Qiu, and Ruibin Bai. A multi-agent framework powered by large language models for automatic heuristic design. *Swarm and Evolutionary Computation*, 107:102487, 2026b. doi: 10.1016/j.swevo.2026.102487.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent–computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024. doi: 10.52202/079017-1601.

Xianliang Yang, Ling Zhang, Haolong Qian, Lei Song, and Jiang Bian. HeurAgenix: Leveraging LLMs for solving complex combinatorial optimization challenges, 2025. URL https://arxiv.org/abs/2506.15196.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills, 2026c. URL https://arxiv.org/abs/2605.23904.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Danial Yazdani, Mohammad Nabi Omidvar, Yuan Sun, Maksud Ibrahimov, and Xiaodong Li. ATLAS: Scaffold-free algorithm synthesis by LLMs via embedding-guided quality-diversity search, 2026. URL https://arxiv.org/abs/2608.15546.

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, pp. 43571–43608. Curran Associates, Inc., 2024. doi: 10.52202/079017-1381.

Davide Zago, André Hottung, Fynn Martin Gilbert, Rossella Cancelliere, and Kevin Tierney. Reinforcement learning guided neural deconstruction search for flexible job scheduling. *Machine Learning*, 115:180, 2026. doi: 10.1007/s10994-026-07116-9.

Pengwei Zhan, Zhen Xu, Qian Tan, Jie Song, and Ru Xie. Unveiling the lexical sensitivity of LLMs: Combinatorial optimization for prompt enhancement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5128–5154. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.295.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. CoEvoSkills: Self-evolving agent skills via co-evolutionary verification, 2026a. URL https://arxiv.org/abs/2604.01687. Accepted to COLM 2026.

Ni Zhang, Zhiguang Cao, Jianan Zhou, Cong Zhang, and Yew-Soon Ong. An agentic framework with LLMs for solving complex vehicle routing problems. In *International Conference on Learning Representations*, 2026b.

Ruiqing Zhao, Rui Liu, Yuan Zuo, Huarong Zhang, Xiao Han, and Junjie Wu. OptiSkill: A hierarchical and evolving skillbank for LLM-based optimization modeling, 2026. URL https://arxiv.org/abs/2609.22987.

## APPENDIX CONTENTS


**A** Detailed Model–Agent Results . . . 14
**B** Runtime Environment . . . 15
**C** Task Definitions, Data Splits, and Evaluators . . . 15
**C.1** Data Splits and Benchmark Aggregation . . . 15
**C.2** Task-Specific Problem Contracts . . . 16
**C.3** Frozen Reference Values and Instance-Level Metrics . . . 16
**D** Round-by-Round Skill Evolution . . . 17
**D.1** Our Skill Evolution . . . 17
**D.2** SkillOpt Skill Evolution . . . 18
**E** Artifact Availability and Execution Templates . . . 18
**E.1** Anonymous Skill Artifacts . . . 18
**E.2** Problem-Independent Execution Templates . . . 18


## A DETAILED MODEL–AGENT RESULTS

Table 2 summarizes performance by COP and includes Initial Skill under the main configuration as a reference. Table 4 reports the corresponding size-, shape-, and class-level results for Evolved Skill across the three model–Agent configurations.

Table 4: Detailed Solver quality of Evolved Skill across model–Agent configurations.

| Benchmark | Scale / class | GPT 5.6-Sol Codex | GPT 5.6-Sol Pi Agent | DeepSeek V4-Flash Codex |
|---|---|---|---|---|
| **TSP — TSPLIB EUC_2D** (node scale) | | | | |
| TSPLIB | 50–500 | 0.02 | 0.02 | 0.09 |
| | 501–1,000 | 0.36 | 0.36 | 0.53 |
| | 1,001–2,000 | 0.63 | 0.63 | 0.94 |
| | 2,001–5,000 | 1.52 | 1.37 | 1.54 |
| | 5,001–20,000 | 2.43 | 2.75 | 2.39 |
| *Average* | | 0.52 | 0.54 | 0.63 |
| **CVRP — CVRPLIB-X** (customer scale) | | | | |
| X | 100–200 | 0.52 | 0.81 | 1.92 |
| | 201–500 | 1.33 | 1.64 | 2.89 |
| | 501–1,000 | 2.00 | 2.59 | 3.46 |
| *Average* | | 1.37 | 1.76 | 2.86 |
| **JSSP — Taillard and DMU** (jobs × machines) | | | | |
| Taillard | $15 \times 15$ | 0.84 | 1.15 | 1.15 |
| | $20 \times 15$ | 2.22 | 2.66 | 3.36 |
| | $20 \times 20$ | 1.86 | 2.60 | 2.94 |
| | $30 \times 15$ | 2.20 | 2.84 | 3.04 |
| | $30 \times 20$ | 3.91 | 5.22 | 5.82 |
| | $50 \times 15$ | 0.03 | 0.04 | 0.17 |
| | $50 \times 20$ | 0.53 | 1.40 | 1.89 |
| | $100 \times 20$ | 0.00 | 0.04 | 0.48 |
| DMU | $20 \times 15$ | 3.63 | 5.17 | 5.47 |
| | $20 \times 20$ | 3.47 | 6.30 | 4.88 |
| | $30 \times 15$ | 4.76 | 7.60 | 8.19 |
| | $30 \times 20$ | 7.09 | 7.98 | 9.73 |
| | $40 \times 15$ | 4.70 | 6.56 | 8.17 |
| | $40 \times 20$ | 6.90 | 8.29 | 11.63 |
| | $50 \times 15$ | 5.45 | 6.30 | 10.24 |
| | $50 \times 20$ | 8.03 | 8.75 | 13.79 |
| *Average* | | 3.48 | 4.56 | 5.68 |
| **3D-BPP — MPV320** (mean bins by class; mean total bins across runs in the summary row ↓) | | | | |
| MPV | 1 | 31.88 | 31.89 | 32.37 |
| | 2 | 31.39 | 31.37 | 32.01 |
| | 3 | 31.65 | 31.66 | 32.23 |
| | 4 | 73.50 | 73.50 | 73.50 |
| | 5 | 17.60 | 17.60 | 17.94 |
| | 6 | 23.76 | 23.76 | 24.06 |
| | 7 | 14.59 | 14.62 | 14.93 |
| | 8 | 20.29 | 20.30 | 20.71 |
| *Mean total bins* | | 9786.3 | 9787.7 | 9909.7 |

# B Runtime Environment

Table 5: Runtime environment versions.

| Component | Version |
|---|---|
| Model | GPT 5.6-Sol |
| Agent harness | Codex 0.155.0-alpha.9.2 |
| Language runtime | Python 3.13.9 |
| NumPy | 2.3.5 |
| Numba | 0.62.1 |
| SciPy | 1.16.3 |

# C Task Definitions, Data Splits, and Evaluators

## C.1 Data Splits and Benchmark Aggregation

Each task has physically separated Train and Test partitions. All Train instances are synthetic. The Solver Agent can read only the task definition and Train. The evaluator selects and executes the external partition; instance identities and frozen reference values never enter the Agent context. Train and external partitions use disjoint generation seeds or public instances. For TSP, the physical Test partition serves as the evolution and selection split $D_{\text{evo}}$, and its instance-level evaluation is returned to Skill Evolver. The Test partitions of CVRP, JSSP, and 3D-BPP are used only for final evaluation of the frozen Skill.

Table 6: Data composition across COPs.

| Task | Train | Test |
|---|---|---|
| TSP | 42 synthetic instances spanning six sizes and seven spatial structures, $n \in \{50, 100, 200, 500, 1000, 10000\}$. | 77 TSPLIB EUC_2D instances, $51 \leq n \leq 18512$ (Reinelt, 1991). |
| CVRP | 40 X-generator shadow instances and 10 structured extensions, for 50 synthetic instances in total; $n \in \{50, 100, 200, 500, 1000\}$. | 100 CVRPLIB-X instances (Uchoa et al., 2017). |
| JSSP | 16 synthetic instances from each of Taillard-exact, DMU-independent, and DMU-calibrated-correlated, for 48 instances in total. | 80 Taillard and 80 DMU public instances, spanning $15\times15$ to $100\times20$ (Taillard, 1993; Demirkol et al., 1998). |
| 3D-BPP | Eight MPV classes $\times$ four item counts $\times$ two difficulty samples, for 64 synthetic instances. | MPV320: eight classes $\times$ four item counts $\times$ ten instances, for 320 benchmark instances; $n \in \{50, 100, 150, 200\}$ (Martello et al., 2000; Gonçalves & Resende, 2013). |

The main table selects literature baselines by benchmark coverage and retains their published values. TSP covers all 77 TSPLIB EUC_2D instances; the ACO-enhanced ReEvo (Ye et al., 2024) and DRAGON use the five size groups of Chen et al. (2026c). CVRP covers all 100 CVRPLIB-X instances; ReEvo (Ye et al., 2024) and AFL come from Table 6 of Zhang et al. (2026b), and their overall averages weight the three groups by 22, 46, and 32 instances. NDS-S (Zago et al., 2026) and SLIM ($GM_{512}$) (Corsini et al., 2024) both cover all 160 Taillard and DMU instances; the combined SLIM average gives equal weight to the two 80-instance families. For 3D-BPP, we use MPV320, the standard 320-instance benchmark introduced by Martello, Pisinger, and Vigo and subsequently used as a common test set: eight classes, four sizes, and ten instances per class–size cell (Martello et al., 2000; Gonçalves & Resende, 2013). The reported literature methods cover the same 320-instance set (Mahvash et al., 2018; Gonçalves & Resende, 2013; Feng et al., 2015). TSP, CVRP, and JSSP averages aggregate all covered instances rather than taking an unweighted mean of the displayed groups; the 3D-BPP summary reports total bins over all MPV320 instances.

The synthetic Train sets span the principal size and structural regimes represented by the public benchmarks. The CVRP set retains the core distributions of the X generator while varying route

length, clustering, and demand structure. The JSSP set covers Taillard-style uniform processing times, DMU-style independent sampling, and correlated DMU structure. MPV320 is procedurally generated, but it is an established public benchmark rather than a Test set generated for this study (Martello et al., 2000; Gonçalves & Resende, 2013).

### C.2 Task-Specific Problem Contracts

Appendix E.2 gives the shared problem-definition interface. Here we record only how the four COPs instantiate that interface. In every task, the evaluator independently checks feasibility and recomputes the minimized objective; final evaluation uses the full per-instance budget.

**TSP.** Instances are symmetric EUC_2D TSPs with 51–18,512 nodes and TSPLIB integer-rounded Euclidean distances. The Solver outputs a permutation of all original node IDs; the evaluator checks exact coverage, closes the tour, and recomputes its length. The time budget increases with instance size from 10 s for $n \leq 100$ to 150 s for $n > 10{,}000$ (intermediate tiers: 15, 30, 45, 60, 90, and 120 s).

**CVRP.** Instances contain one depot, 100–1,000 customers, identical vehicle capacities, and an unlimited fleet; distances use the same integer-rounded Euclidean rule as TSP. The Solver outputs nonempty customer sequences for each route. The evaluator requires every customer to appear exactly once, verifies route capacities, and recomputes total route length. The size-dependent budget ranges from 20 s for $n < 200$ to 150 s for $n = 1{,}000$, with intermediate limits of 30, 45, and 90 s.

**JSSP.** Instances contain 15–100 jobs and 15 or 20 machines. Each job visits every machine exactly once, and all processing times are positive integers. The Solver assigns a nonnegative integer start time to every operation; the evaluator checks job precedence and machine non-overlap before recomputing makespan. The official budgets are 60 s for $15 \times 15$ and $20 \times 15$, 180 s for $100 \times 20$, and 120 s for the remaining tested sizes.

**3D-BPP.** MPV320 instances contain 50–200 integer-dimension cuboids and an unlimited supply of identical bins. Item orientation is fixed. The Solver assigns each item to a consecutively numbered bin and an integer $(x, y, z)$ position; the evaluator checks complete assignment, containment, and pairwise non-overlap, then counts the used bins. The time limit is $T(n) = 60 + 120((n-50)/150)^{1.4}$ seconds.

### C.3 Frozen Reference Values and Instance-Level Metrics

Each Test reference value is frozen before experimentation. TSP contains 77 TSPLIB instances with proven optima. CVRP contains 100 CVRPLIB-X instances evaluated against frozen best-known upper bounds (BKS). JSSP contains 103 closed and 57 open instances. MPV320 contains 21 closed and 299 open instances. For the TSP, CVRP, and JSSP minimization objectives, the instance-level metric is

$$g(x) = \frac{z(x) - \bar{z}(x)}{\bar{z}(x)}, \tag{2}$$

where $z(x)$ is the objective recomputed by the evaluator and $\bar{z}(x)$ is the frozen optimum or BKS. If a Solver improves the frozen upper bound on an open instance, then $g(x) < 0$. JSSP also retains the certified gap to a valid lower bound as a diagnostic metric. The main 3D-BPP table reports the mean number of bins by MPV320 class. Within each development run, the summary metric is the integer total over all 320 instances; the main table then reports the mean of these three run-level totals. Gaps to a reference value or lower bound are used only for diagnosis.

# D ROUND-BY-ROUND SKILL EVOLUTION

Tables 7 and 8 reorganize the two TSP records behind Figure 3. Both trajectories selected versions on the same 77 TSPLIB Test instances. Each gap is the single-run Test mean of the round's candidate Solver, not necessarily the score of the Skill retained after that round. Both runs began at 1.69%. Our retained sequence was Rounds 0, 1, 2, 4, 5, 7, and 8; SkillOpt retained V0, V1, V2, V3, V4, V6, V7, and V9. Rejected candidates were not inherited by the next round. The records describe the proposed revisions and observed outcomes, but do not isolate the causal contribution of an individual Skill rule.

## D.1 OUR SKILL EVOLUTION

Table 7: Our TSP Skill evolution, Rounds 0–10. Gaps are single-run Test means for the candidate in each round; rejected candidates do not enter the retained Skill.

| Round | Candidate gap and decision | Test weakness and Solver design | Recorded Agent decision and Skill-guidance gap | Candidate Skill revision |
|---|---|---|---|---|
| 0 | 1.69%; baseline retained | Feasible construction and small local moves stagnated early in strongly coupled regions; the same search workflow served all instance types. | The Agent extended local search after inspecting sizes and quality. Initial guidance listed features but did not require a testable feature-to-capability link. | No edit; Initial Skill established the baseline. |
| 1 | 1.3684%; accepted | Highly coupled regions inflated tail gap; one construction path produced similar weak starting solutions. | The Agent related input features to search needs. The Skill had not required evidence for feature–mechanism fit or stress tests of valid feature combinations. | Require a computable feature, its implied capability, predicted benefit region, and counterexamples; check batch statistics and valid stress cases. |
| 2 | 1.251329%; accepted | Search improved late in the budget, but candidate-evaluation throughput was low; the Solver accelerated existing moves through incremental evaluation and caching. | The Agent compared short and standard-budget trajectories. The Skill distinguished computation from search qualitatively but gave no explicit diagnostic probe. | Test throughput first when late improvement persists; test search reach when early stagnation survives more computation of the same kind. |
| 3 | 1.461611%; rejected | Additional low-cost channels had different starts but heavily overlapping effective moves, consuming exploration budget. | The Agent treated different starting paths as complementarity. The Skill did not yet require novelty and return per CPU unit to establish channel value. | Try lightweight multi-channel exploration under uncertainty. The candidate worsened gap and was not inherited. |
| 4 | 1.129944%; accepted | More of the same local search reached a quality plateau in coupled cases; partial teardown and reconstruction added joint structural moves. | The Agent compared extended search with short reconstruction probes. The Skill allowed larger neighborhoods but did not require a falsifiable capability hypothesis. | State trigger signals, newly reachable moves, expected benefit, and disconfirming outcomes; begin on a small feature panel. |
| 5 | 1.001970%; accepted | Mean improvement could conceal persistent weak regions; feature-conditioned construction and reconstruction budgets targeted them. | The Agent compared paired runs by feature region and tail quality. Initial guidance did not make stable weak-region gains and absence of comparable new regressions explicit. | Promote an internal Solver version only with preserved correctness, reliable target-region gain, interpretable contribution, and no equally severe new tail loss. |
| 6 | 1.08943%; rejected | A hard feature threshold routed reconstruction unstably near its boundary and in sparsely covered regions. | The Agent turned a correlation into a hard switch without sufficient counterexamples. The Skill warned against narrow thresholds but lacked a boundary-test rule. | Fit a normalized hard boundary and test both sides and unseen scales. The candidate was rejected. |
| 7 | 0.772967%; accepted | Fixed budgets overfunded low-return channels and risked neglecting complementary channels with delayed gains. | The Agent used online CPU, feasible-candidate, gain, and novelty signals. Initial guidance lacked an operational return measure and exploration protection. | Allocate by gain per CPU unit, novelty, and stagnation duration; do not eliminate a unique channel solely for short-term zero return. |
| 8 | 0.44%; accepted | Despite improved scheduling, existing channels still reached the same structural plateau; a structured large neighborhood or local subproblem added different moves. | The Agent checked candidate coverage, improvement structure, and return before integration. The Skill did not require evidence of distinct reachable moves. | Test reach with a prototype, then a feature-diverse panel, then full evaluation; reject mechanisms that repeat existing effective moves. |
| 9 | 0.7159%; rejected | A stronger lower bound consumed search time without reliably changing decisions or improving the final solution. | The Agent treated better diagnostic information as a quality mechanism. The Skill lacked an explicit test of the bound's runtime value. | Reserve budget to strengthen an online lower bound when it indicates headroom. The candidate was rejected. |
| 10 | 0.582396%; rejected | A costly large neighborhood had no gain in a short window, but pruning it reduced rare large improvements. | The Agent used recent return alone. The Skill lacked a sufficient test for distinguishing temporary silence from long-term ineffectiveness. | Reduce a channel to minimum exploration after one unproductive window. The candidate was rejected and Round 8 remained best. |

### D.2 SKILLOPT SKILL EVOLUTION

SkillOpt used complete-rewrite candidates and selected versions by mean gap. Local additions, replacements, and deletions describe the sentence-level review diff within each complete candidate, not a restriction on the optimizer's edit size. Counts show old-to-new declarative sentences and the net change.

Table 8: SkillOpt TSP Skill evolution, Rounds 0–10.

| Round | Input | Candidate gap | Gate; retained version | Method, sentence count, and edit | Rationale |
|---|---|---|---|---|---|
| 0 | — | 1.69% | Baseline; V0 | No candidate edit. | The Initial Skill establishes the common baseline. |
| 1 | V0 | 1.496246% | Accept; V1 | Local addition, $0 \rightarrow 1$ sentence (net $+1$): group Train instances by computable size and structural features and track recurring weak regions. | Overall quality can hide stable weak regions; grouped analysis identifies where improvement is needed. |
| 2 | V1 | 1.347744% | Accept; V2 | Local replacement, $1 \rightarrow 2$ (net $+1$): diagnose early stagnation with improvement curves and neighborhood success before comparing search-scope mechanisms. | Extending the same search may not resolve stagnation; diagnosis should precede a mechanism change. |
| 3 | V2 | 1.333396% | Accept; V3 | Local addition, $0 \rightarrow 2$ (net $+2$): predict target feature regions and use matched-budget on/off comparisons. | Concurrent mechanism and parameter changes make small gains hard to attribute; predictions and ablations sharpen the comparison. |
| 4 | V3 | 1.196246% | Accept; V4 | Local replacement, $1 \rightarrow 1$ (net 0): allocate channel budgets by recent effective gain per unit time while preserving weak-region exploration. | Static resource splits reduce effective throughput; marginal gains offer a better allocation signal. |
| 5 | V4 | 1.298792% | Reject; V4 | Local addition, $0 \rightarrow 1$ (net $+1$): give the leading channel a fixed main budget and minimize all others. | An average lead need not generalize across weak regions; this candidate suppressed complementary channels and regressed. |
| 6 | V4 | 1.089752% | Accept; V6 | Local addition, $0 \rightarrow 1$ (net $+1$): require paired retests of persistent weak regions under the same budget. | Overall averages can conceal continued regional regression, which needs a separate check. |
| 7 | V6 | 1.067873% | Accept; V7 | Local replacement, $1 \rightarrow 1$ (net 0): normalize necessary fixed thresholds and test nearby values across scales. | A fixed threshold can change behavior discontinuously at different instance scales. |
| 8 | V7 | 1.270322% | Reject; V7 | Local deletion, $1 \rightarrow 0$ (net $-1$): remove the full-Train validation requirement after small tests. | A small sample may miss regressions elsewhere; deleting the full validation requirement worsened the candidate. |
| 9 | V7 | 1.01% | Accept; V9 | Local replacement, $1 \rightarrow 2$ (net $+1$): revisit structure after renewed stagnation and admit only mechanisms that change target weak-region search behavior. | After modest gains, a renewed plateau calls for a clear trigger to reconsider the architecture. |
| 10 | V9 | 1.2465% | Reject; V9 | Local addition, $0 \rightarrow 1$ (net $+1$): stop full-budget evaluation of costly mechanisms after no small-sample average gain. | Prematurely dropping expensive mechanisms can remove effective weak-region search; the candidate regressed. |

## E ARTIFACT AVAILABILITY AND EXECUTION TEMPLATES

### E.1 ANONYMOUS SKILL ARTIFACTS

The anonymous repository for the complete Initial Solving Skill, SkillOpt-evolved Solving Skill, our final Solving Skill, and fixed Skill Evolver is `https://anonymous.4open.science/r/ASECO-C5D7/`.

### E.2 PROBLEM-INDEPENDENT EXECUTION TEMPLATES

The five templates below specify the complete execution interface and use angle-bracketed placeholders. They are instantiated in the following order: problem definition, Solver-run configuration, Skill-evolution configuration, Solver-run prompt, and Skill-evolution prompt. Each placeholder is replaced before execution; no source-COP identity is encoded in the templates.

**Solver-run configuration.**

```
{
```

```
  "problem": "<PROBLEM_DEFINITION>",
  "train": {
    "instances": "<TRAIN_INSTANCES>",
    "targets": "<TRAIN_TARGETS>",
    "references": "<OPTIONAL_TRAIN_SOLUTIONS>"
  }
}
```

**Skill-evolution configuration.**

```
{
  "evolution_skill": "<SKILL_EVOLVER>",
  "initial_solver_skill": "<INITIAL_SOLVER_SKILL>",
  "solver_run_config": "<SOLVER_RUN_CONFIG>",
  "solver_run_prompt": "<SOLVER_RUN_PROMPT>",
  "problem": "<PROBLEM_DEFINITION>",
  "workspace": "<EVOLUTION_WORKSPACE>",
  "test": {
    "instances": "<EVOLUTION_INSTANCES>",
    "targets": "<EVOLUTION_TARGETS>"
  },
  "rounds": 10
}
```

**Problem-definition interface.**

````
---
id: <PROBLEM_ID>
name: <PROBLEM_NAME>
objective: <MINIMIZE_OR_MAXIMIZE>
quality_field: <QUALITY_METRIC>
instance_extensions: [<INSTANCE_EXTENSION>]
---

# Problem Definition

State the optimization problem in mathematical or unambiguous operational terms. Define the
decision variables, feasibility conditions, and objective without referring to a particular
dataset or directory.

# Evaluation Domain

Describe the instance family covered by the evaluation. Specify supported sizes, data
types, conventions, and excluded variants. The description must be sufficient to determine
whether an instance belongs to the intended domain.

# Objective and Evaluation Rules

Define whether lower or higher values are better. State how the evaluator validates
feasibility, recomputes the objective, and derives the reported quality metric.

```text
<QUALITY_METRIC_DEFINITION>
```

Explain any reference values, aggregation rules, tolerances, or tie-breaking rules used by
the evaluator.

# Input Contract

Describe the instance format and every field that the Solver may read. Identify stable
instance IDs and any numerical conventions required for parsing.

# Output Contract

Provide the exact machine-readable output schema. Replace the example below with the fields
required by the problem.

```json
{
  "instance": "<INSTANCE_ID>",
  "solution": "<PROBLEM_SPECIFIC_SOLUTION>"
}
```

State the type, range, and feasibility requirements of every field. List all prohibited
fields or auxiliary outputs that the evaluator rejects.
````

```
# Constraints and Resource Limits

Specify the allowed language and dependencies, memory limit, per-instance time limit,
parallelism policy, and any prohibited external tools. If budgets vary by instance size,
define the complete size-to-budget mapping here.
```

**Solver-run prompt.**

```
Read the Solver-development Skill:

`<SOLVER_SKILL>`

Read the Solver-run configuration:

`<SOLVER_RUN_CONFIG>`

Develop, validate, and deliver the Solver according to the Skill, problem contract, and
configured information boundary.
```

**Skill-evolution prompt.**

```
Read the Skill Evolver:

`<SKILL_EVOLVER>`

Read the Skill-evolution configuration:

`<SKILL_EVOLUTION_CONFIG>`

Complete the configured evolution rounds. In each round, diagnose the current
Solver-quality limitation, derive a problem-independent revision direction, and evaluate
the candidate Skill through a fresh Solver-development run.
```