EvoORSkill: When Optimization Skills Evolve, Who Checks the Math?

Anonymous authors
Paper under double-blind review

Abstract

Large language models can translate operational requirements into optimization models and solver code, but code that runs successfully may still solve the wrong decision problem. Reusing generated procedures reduces repeated modeling work, yet adapting them to new requirements can change which decisions the model allows without causing execution errors. We introduce EvoORSkill, an agent framework that evolves reusable operations research skills together with their revision strategies, without retraining the underlying LLM. Each skill combines an executable modeling procedure with explicit mathematical relationships, applicability conditions, and version information. EvoORSkill organizes these skills in an evolving skill hypergraph (ESH), which connects the procedures assembled for a task through the mathematical objects they share. When the agent proposes a revision in response to task feedback, ESH traces the affected relationships in both the accepted and candidate hypergraphs. Specification, requirement, and retention checks then determine whether the revised skills can enter the reusable library. Historical replay compares revision strategies using records of past attempts with compatible versions, avoiding repeated candidate generation and evaluation. The selected strategy guides further skill revisions, whose outcomes extend the history used to refine the strategy. Across 518 problems from six datasets, EvoORSkill achieves 81.5% average solution accuracy, exceeding the strongest evaluated baseline by 5.0 percentage points. Removing cross-skill dependency checking or specification and requirement checks reduces accuracy by 6.6 and 13.3 percentage points, respectively. Historical replay reduces library-building tokens by 14.8% compared with generating and evaluating candidates afresh. With its evolved library frozen, EvoORSkill achieves 57.0% accuracy on 100 OptMATH problems, outperforming the same agent without a library by 25 percentage points while using 28.3% fewer online tokens.

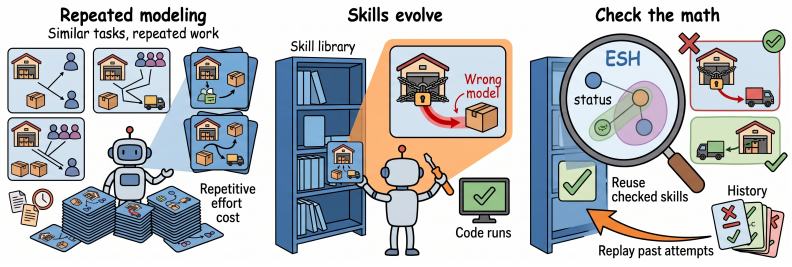

Figure 1: **Problem setting and EvoORSkill overview.** Reuse creates a verification problem: revised code may still execute after its feasible set has drifted from the request. EvoORSkill addresses this at two levels: ESH turns shared mathematical objects into revision obligations, while historical replay reuses version-compatible evidence to improve subsequent revisions.

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.

1 INTRODUCTION

An optimization model can execute successfully while solving the wrong problem. Operations research (OR) turns operational requirements into objectives and constraints for decisions in logistics, manufacturing, healthcare, and other resource-constrained settings. Solvers can optimize these models reliably, but constructing the right model and its executable code still requires specialist expertise (Ahmaditeshnizi et al., 2024). Large language models (LLMs) can interpret requirements, generate solver programs, and coordinate execution (Gao et al., 2023; Wang et al., 2024); benchmarks such as NL4Opt and Mamo now evaluate this translation systematically (Ramamonjison et al., 2022; Huang et al., 2025b). The remaining difficulty is not execution alone, but preserving the intended decision problem as generated procedures are reused and changed.

Existing work focuses on different parts of this pipeline. ORLM and LLaMoCo train models for optimization modeling, while OptMATH supplies synthetic training and evaluation data (Huang et al., 2025a; Ma et al., 2026; Lu et al., 2025). At inference time, OptiMUS revises formulations and code through execution feedback, Chain-of-Experts coordinates specialized agents, and LLMOPT combines a structured formulation with self-correction (Ahmaditeshnizi et al., 2024; Xiao et al., 2024; Jiang et al., 2025). Reuse systems retain more of this work: OptSkills distills optimization procedures, AlphaOPT updates an experience library, and ACE evolves reusable guidance (Yang et al., 2026a; Kong et al., 2025; Zhang et al., 2026). Yet reuse introduces a different failure mode. Replacing an "exactly once" constraint with "at most once," for example, leaves executable code but permits customers to remain unserved. **Once a modeling procedure is reused, the central problem becomes controlled revision: a local edit must reveal which mathematical relationships can no longer be trusted.**

We address this problem with EvoORSkill, an agent framework that evolves reusable OR modeling skills without retraining the underlying LLM. Each skill stores an executable procedure together with its declared mathematical relationships, applicability conditions, and version. EvoORSkill organizes these skills through an evolving skill hypergraph (ESH), whose nodes are typed mathematical objects and whose hyperedges are skill versions. Dependencies therefore arise from the variables and parameters shared in an assembled model rather than from a separately declared workflow graph. Task feedback proposes a candidate revision; ESH traces its effects through the accepted and candidate models, after which specification, requirement, and retention checks determine whether the candidate may enter the reusable library. **Within EvoORSkill, ESH turns a local edit into a finite set of requirement-grounded obligations that must be discharged before reuse.** This mechanism extends skill and workflow evolution to the mathematical models that agent systems construct (Huang et al., 2026).

Candidate admission controls what may be reused, but it does not determine which skill to revise next or how much evaluation budget to spend. Reconsidering these choices through fresh LLM calls would repeatedly generate and test candidates whose outcomes are already known. EvoORSkill therefore records each revision, its affected relationships, check outcomes, task accuracy, and token cost. Historical replay exposes compatible records to candidate revision strategies, compares the skill collections they would visit, and returns to real exploration only when a strategy selects an unsupported action. The resulting loop evolves two coupled objects: task feedback changes the modeling skills, while accumulated revision evidence changes the strategy used to revise them. The base LLM and admission criteria remain fixed throughout this process.

We evaluate EvoORSkill on 518 problems from six datasets. It reaches 81.5% average solution accuracy, 5.0 percentage points above the strongest evaluated baseline. Removing cross-skill dependency checking or the specification and requirement checks lowers accuracy by 6.6 and 13.3 points, respectively, while increasing regression on previously solved problems. Historical replay reduces library-building tokens by 14.8% relative to fresh candidate generation and evaluation. With the evolved library frozen, EvoORSkill reaches 57.0% accuracy on 100 OptMATH problems—25 percentage points above the same agent without a library—and uses 28.3% fewer online tokens.

Our main contributions are as follows:

1. **Joint evolution of skills and revision strategies.** EvoORSkill turns task feedback into versioned OR-skill updates without retraining the underlying LLM, allowing modeling procedures to accumulate across tasks.

2. **Dependency-aware candidate admission.** ESH represents skills as hyperedges over shared mathematical objects, traces the relationships affected by a revision, and admits a candidate only after specification, requirement, and retention checks pass.
3. **Evidence-efficient strategy evolution.** Historical replay compares revision strategies without regenerating and reevaluating recorded candidates; six source datasets and frozen-library transfer evaluate accuracy, retention, cost, and transfer.

2 Related Work

2.1 Automated optimization modeling

LLM-based optimization research develops both modeling ability and the procedures that use it. NL4Opt and Mamo evaluate translation from language to mathematical representations (Ramamonjison et al., 2022; Huang et al., 2025b). ORLM, LLaMoCo, and AutoOR adapt models through training, while OptMATH supplies synthetic modeling data (Huang et al., 2025a; Ma et al., 2026; Motwani et al., 2026; Lu et al., 2025). The training targets include identifying variables, objectives, and constraints, and producing code in the expected format. At inference time, OptiMUS and LLMOPT combine formulation, code generation, and correction; Chain-of-Experts and CP-Agent organize specialized reasoning and solver interaction (Ahmaditeshnizi et al., 2024; Jiang et al., 2025; Xiao et al., 2024; Szeider, 2026). OptiTree uses tree search to decompose problems and combine modeling ideas (Liu et al., 2025). Across these approaches, the main outputs are a mathematical formulation and code that a solver can execute. Reuse extends this work across tasks: OptSkills distills optimization procedures, and AlphaOPT updates an experience library (Yang et al., 2026a; Kong et al., 2025). Retaining a successful procedure does not establish that later revisions still express the intended problem. By keeping code together with its mathematical relationships and applicability conditions, EvoORSkill gives each evolving skill an explicit basis for checking its continued use. These lightweight skills also supply OR-specific procedures without adapting the underlying model weights.

2.2 Evolving skills and revision strategies

Experience can be retained as instructions, programs, or structured records. Reflexion and ACE revise guidance from feedback, while Trace2Skill and SkillOpt extract or update reusable skills (Shinn et al., 2023; Zhang et al., 2026; Ni et al., 2026; Yang et al., 2026b). MemGovern and MemCoder organize coding experience, and Recuris uses verified task progress to guide memory evolution (Wang et al., 2026; Deng et al., 2026; Yu et al., 2026). Structure determines how this experience is combined: Graph of Thoughts connects reasoning units, SkillDAG records skill dependencies and conflicts, and EvoOR-Agent co-evolves workflow graphs and reasoning trajectories (Besta et al., 2024; Bai et al., 2026; Huang et al., 2026). Workflow and skill dependencies describe how components interact, but a revision can also alter mathematical relationships across constraints. We introduce the evolving skill hypergraph (ESH) to connect skill evolution with mathematical checking. By making shared mathematical objects explicit, ESH links revisions to the relationships that must be checked against task requirements.

Evolution also involves choosing what to change and where to spend evaluation effort. AutoHarness generates agent control code, while Meta-Harness searches such code using previous evaluations (Lou et al., 2026; Lee et al., 2026). GEPA uses reflection on execution traces to revise prompts, and AdaEvolve adapts exploration and resource allocation using observed improvements (Agrawal et al., 2026; Cemri et al., 2026). Dream-RSI evaluates improvement policies through historical replay (Zheng et al., 2026a). For skill evolution, the cost grows if each strategy comparison regenerates and reevaluates the same candidates. EvoORSkill reuses recorded outcomes and costs as evaluation evidence, allowing strategies to be compared without repeating those candidate trials.

2.3 Execution feedback and mathematical checks

Tool use connects language reasoning to executable actions, as summarized in surveys of LLMs, agents, and tool learning (Zhao et al., 2026; Wang et al., 2024; Qu et al., 2025). PAL and Chain of Code delegate computation to interpreters or code emulation (Gao et al., 2023; Li et al., 2024);

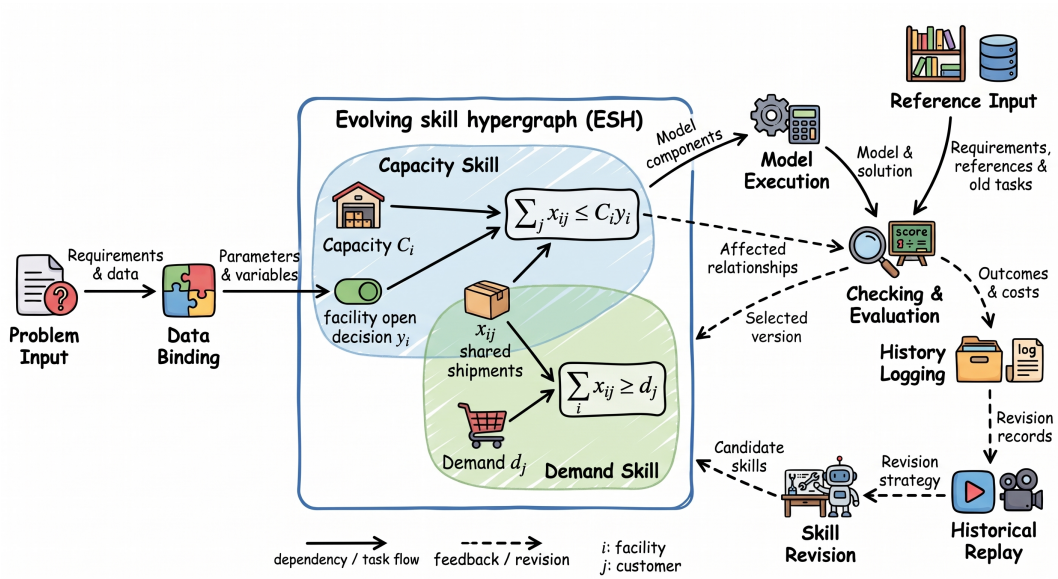

Figure 2: **EvoORSkill links model construction, dependency tracing, candidate checking, and replay.** Overlapping hyperedges expose capacity and demand skills coupled by shipment variable x_{ij}; dashed arrows carry evaluation evidence back to revision and replay. Here C_i, y_i, and d_j denote facility capacity, the opening decision, and customer demand.

Toolformer, ToolLLM, API-Bank, and in-context tool-use studies address tool selection and invocation (Schick et al., 2023; Qin et al., 2024; Li et al., 2023; Zheng et al., 2026b). Plan-and-Act, WebAgent, and CCA organize planning and execution across interactive tasks (Erdogan et al., 2025; Gur et al., 2024; Hang et al., 2025). InterCode and execution-guided generation use execution signals during coding; AgentBench and EnvScaler provide environments for evaluating actions (Yang et al., 2023; Lavon et al., 2025; Liu et al., 2024; Song et al., 2026). For optimization, evaluation must also address whether the model expresses the required decisions and constraints. A solver can return a feasible solution to an incorrectly specified model. Successful execution, feasibility within that model, and agreement with the task therefore require different checks.

Design by contract makes input and output requirements explicit (Meyer, 1992). Mutation testing probes checks with injected faults, while metamorphic testing examines expected relationships between transformed cases (Jia & Harman, 2011; Chen et al., 2018). Applying these testing principles to evolving skills requires checks that stay tied to the intended problem as procedures change. Within ESH, these checks follow the mathematical relationships affected by revisions, using reference answers or independent models and previously solved tasks to assess the revised model and its effects on existing capabilities.

3 EvoORSkill

EvoORSkill uses an LLM-based agent to develop OR capabilities by updating executable skills within an evolving skill hypergraph (ESH), which explicitly represents the mathematical objects and relationships used by those skills. Given a task, it binds source data to suitable skills, constructs an optimization model, and evaluates the solver output. When evaluation exposes modeling errors or requirements that existing skills cannot handle, EvoORSkill uses the feedback to revise those skills or add new ones. ESH makes the relationships affected by a revision explicit, so checks can follow its effects across skills. Accepted versions support subsequent tasks, while recorded attempts let historical replay compare revision strategies. Figure 2 shows the task flow through ESH and the feedback loop linking evaluation, history logging, replay, and skill revision.

3.1 VERSIONED MATHEMATICAL SKILLS IN AN EVOLVING HYPERGRAPH

A reusable optimization skill can be revised safely only if the system can identify the mathematical objects that the revision may affect. Consider a delivery task in which facilities have capacities, customers have demands, and opening and shipping incur costs. A capacity relation couples facility capacity C_i, opening decision y_i, and shipment quantities x_{ij} through $\sum_j x_{ij} \le C_i y_i$, whereas a demand relation constrains the same shipments through $\sum_i x_{ij} \ge d_j$. Because both relations act on x_{ij}, changing the capacity relation can alter their joint feasible set even when the demand skill itself is unchanged. The evolving skill hypergraph (ESH) exposes this coupling through the mathematical objects shared by the two skills.

For a task $q = (u_q, d_q, \mathcal{R}_q)$, u_q contains the written requirements, d_q contains the source data, and $\mathcal{R}_q$ contains the requirement records against which skill candidates are evaluated. A record $r = (\phi_r, \sigma_r, \rho_r)$ associates a checkable condition ϕ_r with the mathematical objects σ_r that it concerns and the source ρ_r from which it was obtained. For example, the requirement that a closed facility cannot ship has support $\{y_i, x_{ij}\}$ and condition $y_i = 0 \Rightarrow \sum_j x_{ij} = 0$. EvoORSkill creates these records from available reference specifications or confirmed task interpretations before proposing a candidate revision. The records remain fixed while the candidate is evaluated; changed task requirements create a new record version. Data binding maps source entities to typed parameters and variables while checking identifiers, indices, units, and domains. A missing or incompatible binding prevents the corresponding skill from being used.

ESH stores these typed mathematical objects as nodes and each versioned skill as a directed hyperedge:

$$\mathcal{H}_t = (\mathcal{V}_t, \mathcal{E}_t), \qquad e_s = (I_s \to O_s;\, g_s, p_s, a_s, \nu_s). \tag{1}$$

Here $\mathcal{H}_t$ is the accepted hypergraph after t update rounds. The sets $I_s, O_s \subseteq \mathcal{V}_t$ identify the objects read and produced by skill s; g_s states its mathematical relations, p_s is the executable procedure that constructs them, a_s specifies when the skill applies, and ν_s identifies its version. Because one relation may couple several parameters and variables, a skill forms a hyperedge rather than a pairwise link. Let $\mathrm{supp}(s) = I_s \cup O_s$. Two skills are mathematically coupled when their supports intersect; in the delivery example, the capacity and demand skills are coupled through x_{ij}. Their dependency is therefore induced by the assembled model rather than declared independently of it.

Given a valid binding b_q, applicability and data checks select $\mathcal{S}_t(q)$. Its procedures build the executable model, while their declared relations form its specification:

$$M_t(q) = \bigoplus_{s \in \mathcal{S}_t(q)} p_s(b_q), \qquad \Phi_t(q) = \bigwedge_{s \in \mathcal{S}_t(q)} g_s(b_q). \tag{2}$$

The operator $\bigoplus$ merges the variables, constraints, and objective terms emitted by the selected procedures and is defined only when shared objects agree in type, domain, and index set; $\Phi_t(q)$ collects their declared mathematical relations. The pair (p_s, g_s) is therefore an executable–declarative contract: one constructs the model fragment, while the other states what that fragment must mean. Shared support identifies which contracts require rechecking. When no existing skill applies, the agent proposes a new hyperedge, which remains outside $\mathcal{H}_t$ until it passes the same revision checks.

3.2 REVISION IMPACT AND CANDIDATE ADMISSION

Task feedback may reveal that the current capacity skill omits the opening decision y_i, allowing a closed facility to ship goods. The revision strategy π_t responds with an add or replace operation δ_t, which produces a candidate $\widetilde{\mathcal{H}}_t$ while leaving the accepted hypergraph $\mathcal{H}_t$ unchanged. Write $s \sim s'$ when the supports of two skills intersect, as defined in Section 3.1. Let $\mathcal{H}[q]$ denote the subhypergraph induced by the skills selected for task q, and let $\Delta\mathcal{S}_t$ contain the skills directly added or replaced by δ_t. The affected skills are

$$\widetilde{\mathcal{H}}_t = \mathcal{H}_t \boxplus \delta_t, \qquad \mathcal{I}_t(q, \delta_t) = \mathrm{cl}_{\sim}^{\mathcal{H}_t[q] \cup \widetilde{\mathcal{H}}_t[q]}(\Delta\mathcal{S}_t). \tag{3}$$

The closure follows the support-intersection relation from Section 3.1; it contains every selected skill reachable from $\Delta\mathcal{S}_t$ through shared mathematical objects. Computing it over both task-specific hypergraphs retains couplings removed by the revision and includes couplings newly introduced by it. This construction is conservative: shared support may trigger an unnecessary check, but no dependency visible in either hypergraph is omitted.

The impact set determines which requirement records become obligations. Writing $\mathrm{supp}(\mathcal{I}) = \bigcup_{s\in\mathcal{I}} \mathrm{supp}(s)$, let $\mathcal{R}_t^{\delta}(q)$ contain the records $r \in \mathcal{R}_q$ for which $\sigma_r \cap \mathrm{supp}(\mathcal{I}_t(q,\delta_t)) \neq \varnothing$. For a solver-expressible condition ϕ_r, the candidate model $\widetilde{M}_t(q)$, assembled as in Eq. (2), is checked by searching for a violating feasible solution:

$$\begin{aligned} \mathrm{CE}(\widetilde{M}_t(q), r) &= \{z : z \models \widetilde{M}_t(q) \wedge \neg\phi_r(z)\}, \\ \mathrm{Check}(\widetilde{M}_t(q), r) = \textsc{pass} &\iff \mathrm{CE}(\widetilde{M}_t(q), r) = \varnothing. \end{aligned} \tag{4}$$

Operationally, PASS requires the solver to certify that the counterexample set is empty; a feasible witness returns FAIL, and a timeout or unresolved solve returns UNKNOWN. For the closed-facility requirement, the checker fixes $y_i = 0$ and maximizes $\sum_j x_{ij}$; a value above the preset tolerance is a counterexample. Other conditions use a registered controlled-instance or independent-reference checker. Together, the impact closure and counterexample search give a scoped guarantee: with complete supports and records, every registered requirement touched in either graph is rechecked, and a pass excludes a feasible counterexample to each solver-expressible condition. They do not establish candidate feasibility or recover unrecorded requirements.

Candidate admission therefore combines three checks. The specification check V_{spec} compares the model fragments emitted by impacted procedures with their declared relations g_s. The requirement check V_{req} requires every record in $\mathcal{R}_t^{\delta}(q)$ to pass its registered checker, and the retention check V_{retain} evaluates the designated previously solved tasks. The first two checks aggregate their task-level results over the fixed checking set used for that candidate. Each component returns PASS, FAIL, or UNKNOWN; a candidate is admitted only when all three return PASS. Let $\mathcal{P}_t$ contain the current collection $\mathcal{H}_t$ and every candidate admitted in round t. Version selection is then

$$\mathcal{H}_{t+1} = \operatorname*{argmax}_{\mathrm{lex},\, \mathcal{H}\in\mathcal{P}_t} \big(A(\mathcal{H}), -C(\mathcal{H})\big). \tag{5}$$

All collections in $\mathcal{P}_t$ are evaluated on the same fixed development tasks. The lexicographic rule first maximizes family-balanced solution accuracy A, then minimizes average solving token cost C among equally accurate collections; unsuccessful runs remain in both statistics. If both values tie among the maximizing collections, the current collection is retained when it belongs to the tied set; otherwise, the earliest evaluated candidate is selected. The gate establishes admission only with respect to the recorded requirements, registered checkers, and retained tasks. It is not a proof that the candidate is equivalent to every possible interpretation of the natural-language request. Every attempt and its evidence are stored, including rejected candidates, so that historical replay can account for the gains and costs of the revision strategy that produced it.

3.3 Historical replay for strategy evolution

Each evaluated revision produces an immutable record $h_i = (\mathcal{H}_i, \delta_i, \widetilde{\mathcal{H}}_i, o_i, c_i, \kappa_i)$. The outcome o_i contains the affected relationships, check evidence, and task evaluation; c_i is the generation and evaluation cost. The compatibility key κ_i identifies the exact parent and candidate versions together with the data, requirement, checker, and evaluator versions. A record is replayed only when this key matches the state being reconstructed; otherwise, the candidate requires fresh evaluation.

On a compatible history, a revision strategy observes the same starting collection and selects its next recorded action. Replay reveals the outcome and cost of that action, after which the strategy may continue from the resulting candidate or stop. Outcomes of unselected actions remain hidden. A branch ends when no compatible action remains or the next recorded cost exceeds its remaining budget; an unsupported action returns control to real exploration. Replay therefore preserves the information order of the original revision process without repeating LLM calls or solver runs.

The strategy-revision agent uses one set of histories to propose rules for choosing the next error, stopping a revision sequence, and allocating its remaining budget. The candidate set Π_t contains these proposals and the current strategy π_t. A separate, fixed history set $\mathcal{B}$ evaluates them under the same recorded evidence:

$$\pi_{t+1} = \operatorname*{argmax}_{\pi \in \Pi_t} \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} \big[\Delta A_b(\pi) - \lambda c_b(\pi)\big]. \tag{6}$$

For history b, Eq. (5) selects the best collection among its starting version and the candidates visited by π; $\Delta A_b(\pi)$ is the resulting accuracy gain over that starting version on the fixed development tasks. The normalized cost $c_b(\pi)$ is the recorded generation and evaluation tokens of all visited attempts, including failures, divided by the history budget. The fixed weight $\lambda \geq 0$ determines how much this cost reduces the replay score. Replay overhead and the cost of proposing strategies are logged separately from c_b.

Ties favor the current strategy and then proposal order. If the proposal or evaluation histories are empty, EvoORSkill retains π_t, which is initialized before exploration. The selected strategy governs subsequent real revisions, and their records extend the histories available to later comparisons. Skill versions thus evolve from task feedback, while the strategy that revises them evolves from versioned evidence produced by the same process.

4 Experiments

4.1 Experimental setup

Our experiments examine whether evolving modeling skills improves OR solving, which checks help preserve existing capabilities, and whether replay makes revision more efficient. We first compare solution accuracy across six datasets, then isolate the framework components through ablations. A final test freezes the evolved library before transfer to OptMATH.

All methods in the main comparison use DeepSeek v4 as the base model. We compare EvoORSkill with direct generation, Reflexion, Chain-of-Experts, OptiMUS, ORLM, AlphaOPT, LLMOPT, and OptSkills (Shinn et al., 2023; Xiao et al., 2024; Ahmaditeshnizi et al., 2024; Huang et al., 2025a; Kong et al., 2025; Jiang et al., 2025; Yang et al., 2026a). The source evaluation contains 518 test problems: 100 each from NL4Opt, Mamo Easy, Mamo Complex, NLP4LP, and IndustryOR, and 18 from ComplexOR.

Solution accuracy (SA) measures the percentage of correct answers against benchmark references. Avg. SA averages the six dataset accuracies with equal weight. Regression measures how often library updates turn previously correct answers into incorrect ones. We first identify the test problems solved correctly using the shared initial library $\mathcal{H}_0$. For each configuration, regression is the percentage of this same set answered incorrectly using its final library. These independent test problems are excluded from revision checks and version selection. Build tokens cover library construction and evolution, in millions; online tokens measure average solving cost per problem, in thousands. Ablations share the task sequence and scheduled revision rounds, except that the initial-library configuration stops updating after construction.

4.2 Overall solution accuracy

EvoORSkill achieves the highest average accuracy in Table 1: 81.5%, compared with 76.5% for the strongest baseline, LLMOPT, and 55.6% for direct generation. The gain over LLMOPT is 5.0 percentage points. It leads on NL4Opt and IndustryOR with 96.0% and 78.0%, respectively. On IndustryOR, the next highest score is AlphaOPT at 60.0%, an 18-point difference. EvoORSkill also ties the best results on NLP4LP (87.0%) and ComplexOR (72.2%).

The advantage varies by dataset. LLMOPT scores 97.0% on Mamo Easy, above EvoORSkill at 95.0%; on Mamo Complex, AlphaOPT reaches 76.0% while EvoORSkill reaches 61.0%. EvoORSkill leads or ties on four datasets; its largest deficit is on Mamo Complex. The component comparisons below examine which parts of skill evolution contribute to the overall result.

Table 1: Solution accuracy (SA, %; higher is better). All methods are reproduced using DeepSeek v4 as the base model. Avg. gives equal weight to all six datasets. Bold marks column maxima, including ties.

Method	NL4Opt	Mamo Easy	Mamo Complex	NLP4LP	ComplexOR	IndustryOR	Avg.
Direct generation	64.0	75.0	39.0	68.0	55.6	32.0	55.6
Reflexion	53.0	67.0	51.0	46.0	27.8	34.0	46.5
Chain-of-Experts	64.0	63.0	35.0	53.0	22.2	27.0	44.0
OptiMUS	78.0	71.0	44.0	72.0	66.7	45.0	62.8
ORLM	86.0	82.0	37.0	79.0	50.0	33.0	61.2
AlphaOPT	81.0	83.0	**76.0**	**87.0**	61.1	60.0	74.7
LLMOPT	93.0	**97.0**	68.0	83.0	**72.2**	46.0	76.5
OptSkills	82.0	85.0	63.0	77.0	**72.2**	36.0	69.2
EvoORSkill	**96.0**	95.0	61.0	**87.0**	**72.2**	**78.0**	**81.5**

4.3 Effects of skill evolution, ESH checks, and replay

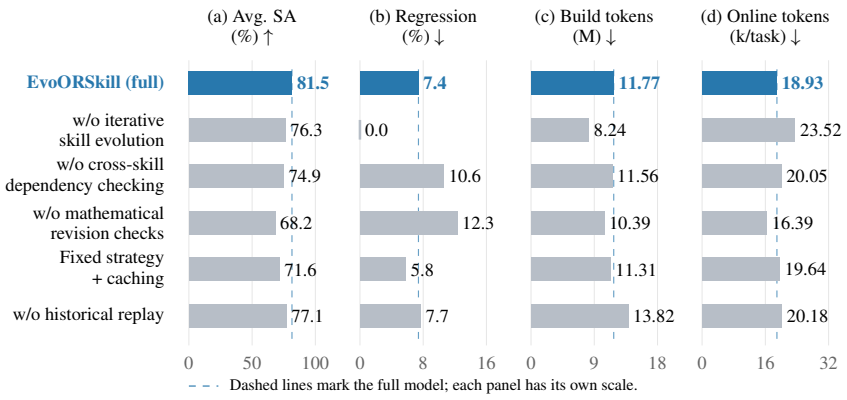

Figure 3: **Ablations separate the effects of evolution, dependency tracing, revision checks, and replay.** Dashed lines mark full EvoORSkill; regression uses the same initially solved subset for every configuration. Each panel has its own zero-based scale.

Continued skill evolution improves solving after library construction. Freezing $\mathcal{H}_0$ lowers Avg. SA by 5.2 points, from 81.5% to 76.3% (Figure 3). The frozen library costs fewer build tokens, 8.24M versus 11.77M, but requires more online tokens, 23.52k versus 18.93k per problem. Its regression is 0.0% because the library remains unchanged. Evolution therefore incurs additional construction cost and some loss of initially solved cases, while increasing overall accuracy and reducing later solving cost.

ESH checks reduce the accuracy loss and regression associated with skill revisions. Removing checks on connected skills, while retaining local checks on the edited skill, lowers SA to 74.9% and raises regression from 7.4% to 10.6%. In a separate ablation, removing mathematical revision checks lowers SA to 68.2% and raises regression to 12.3%; execution checks and final solution evaluation remain active. The two interventions separate the properties formalized in Section 3.2: the first restricts impact tracing to the edited skill, whereas the second removes the obligations applied after tracing. Their 6.6- and 13.3-point accuracy losses support both roles without treating either ablation as a proof of semantic correctness. Omitting mathematical checks uses the fewest online tokens (16.39k), but also has the lowest accuracy among the ablations.

Updating the revision strategy raises overall accuracy relative to a fixed strategy. With a fixed strategy and caching, skills continue to change, but the rule for choosing revisions remains fixed. Avg. SA falls to 71.6%, 9.9 points below the full framework, even though regression is lower at 5.8%. Regression tracks only initially correct problems; overall SA also measures performance on the rest

of the test set. The full framework achieves greater overall accuracy while retaining fewer of those initially correct answers than the fixed strategy.

Historical replay lowers library-building cost while improving accuracy. Without replay, strategies still evolve, but comparing them requires fresh candidate generation and evaluation. This configuration uses 13.82M build tokens and reaches 77.1% SA. Replay reduces build tokens to 11.77M, a reduction of approximately 14.8%, while increasing SA by 4.4 points. Replay reuses recorded outcomes during strategy comparison. The resulting library also uses fewer online tokens: 18.93k versus 20.18k tokens per problem.

4.4 Transfer to OptMATH

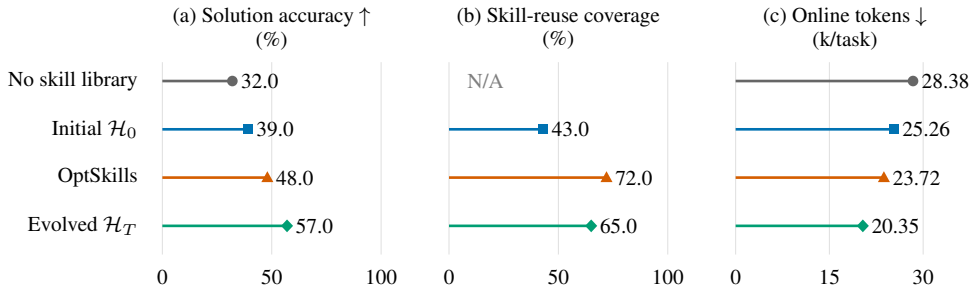

Figure 4: **Frozen-library transfer to 100 OptMATH problems.** The evolved library improves accuracy and online efficiency, whereas reuse coverage alone does not predict success. Source libraries receive no target updates; online cost excludes construction.

We test whether skills evolved on the six source datasets remain useful on 100 OptMATH problems (Lu et al., 2025). The initial library $\mathcal{H}_0$, evolved library $\mathcal{H}_T$, and OptSkills source-built library are frozen, with no updates from target feedback. Reuse coverage is the percentage of target problems whose final candidate model actually uses at least one frozen source modeling skill. Online token counts exclude source-library construction.

The evolved library reaches 57.0% SA (Figure 4), compared with 32.0% without a library, 39.0% with $\mathcal{H}_0$, and 48.0% with OptSkills. The 18-point gain over $\mathcal{H}_0$ shows that source-domain evolution produces skills that help on new tasks after updates stop. Online cost falls to 20.35k tokens per problem, 28.3% below the no-library condition. Coverage increases from 43.0% with $\mathcal{H}_0$ to 65.0% with $\mathcal{H}_T$. OptSkills has higher coverage at 72.0% but lower accuracy at 48.0%. Actual skill use and successful solving are therefore distinct outcomes; the evolved library transfers with higher accuracy despite covering fewer problems than OptSkills.

5 Conclusion

EvoORSkill treats reuse as controlled model revision rather than repeated generation. ESH exposes the mathematical relationships affected by a skill update and gates admission on specification, requirement, and retention checks; historical replay uses recorded attempts to improve the strategy that proposes those updates. Across six datasets, EvoORSkill reaches 81.5% average solution accuracy, 5.0 points above the strongest evaluated baseline. Its frozen library reaches 57.0% on OptMATH, 25 points above the same agent without a library, while using 28.3% fewer online tokens. Ablations further separate the two mechanisms: ESH checks limit regressions from skill changes, whereas replay cuts library-building tokens by 14.8% relative to fresh candidate evaluation. These results support evolvable OR skills whose revisions remain inspectable; extending the retained-task evidence across longer revision sequences is the next step.

AI USE STATEMENT

Generative AI assisted with manuscript drafting, editing and translation, refinement of the conceptual framework and mathematical formulation, experimental design, and interpretation of results. The authors directed these revisions and take responsibility for the final content.

ETHICS STATEMENT

Automated optimization can influence how resources and services are allocated. A mathematically valid solution may still reflect incomplete requirements or overlook the interests of affected people. EvoORSkill checks modeling relationships against specified requirements; decisions about fairness, safety, and acceptable trade-offs therefore need to be expressed in those requirements and reviewed in the application context. For deployment in settings such as logistics or healthcare, domain experts should examine the objectives, constraints, and resulting decisions before they are acted upon. Task data and revision histories should also be handled according to applicable privacy and access requirements, since they may contain sensitive operational information.

REPRODUCIBILITY STATEMENT

Section 3 describes the skill representation, revision checks, version selection, and historical replay used by EvoORSkill. Section 4 defines the evaluation datasets, accuracy and regression metrics, ablation settings, and frozen-library transfer protocol. The accompanying aggregate result files and scripts support regeneration of the reported tables and experimental figures. Dataset-level averages and relative token reductions can be recomputed from these files. Reproducing the underlying agent runs additionally requires the model configuration, task splits, prompts, solver settings, and revision budgets; the aggregate results alone support verification of the reported summaries.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, pp. 8479–8565, 2026.

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.

Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Yang You, and Ivor W. Tsang. SkillDAG: Self-evolving typed skill graphs for LLM skill selection at scale. *arXiv preprint arXiv:2606.03056*, 2026.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024. doi: 10.1609/aaai.v38i16.29720.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. AdaEvolve: Adaptive LLM driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys*, 51(1):4:1–4:27, 2018. doi: 10.1145/3143561.

Yi-Xuan Deng, Xiaoqin Liu, Yi Zhang, Guo-Wei Yang, and Shuojin Yang. Your code agent can grow alongside you with structured memory. *arXiv preprint arXiv:2603.13258*, 2026.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-Act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 15419–15462. PMLR, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world WebAgent with planning, long context understanding, and program synthesis. In *International Conference on Learning Representations*, pp. 52690–52717, 2024.

Tiankai Hang, Shuyang Gu, Dong Chen, Xin Geng, and Baining Guo. CCA: Collaborative competitive agents for image editing. *Frontiers of Computer Science*, 19(11):1911367, 2025. doi: 10.1007/s11704-025-41244-0.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025a. doi: 10.1287/opre.2024.1233.

Jiahao Huang, Peilan Xu, Xiaoya Nan, and Wenjian Luo. Co-evolving agent architectures and interpretable reasoning for automated optimization. *arXiv preprint arXiv:2604.17708*, 2026.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-naacl.146.

Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011. doi: 10.1109/TSE.2010.62.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, pp. 101580–101606, 2025.

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. *arXiv preprint arXiv:2510.18428*, 2025.

Boaz Lavon, Shahar Katz, and Lior Wolf. Execution guided line-by-line code generation. In *Advances in Neural Information Processing Systems*, volume 38, pp. 147615–147640. Curran Associates, Inc., 2025. doi: 10.52202/085713-4936.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. In *Conference on Language Modeling (COLM)*, 2026.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of Code: Reasoning with a language model-augmented code emulator. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28259–28277. PMLR, 2024.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187.

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye Hao. OptiTree: Hierarchical thoughts generation with tree search for LLM optimization modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 120713–120781. Curran Associates, Inc., 2025. doi: 10.52202/085713-4027.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, pp. 52989–53046, 2024.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: Improving LLM agents by automatically synthesizing a code harness. In *ICLR 2026 Workshop on Recursive Self-Improvement*, 2026.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40769–40802. PMLR, 2025.

Zeyuan Ma, Yue-Jiao Gong, Hongshu Guo, Jiacheng Chen, Yining Ma, Zhiguang Cao, and Jun Zhang. LLaMoCo: Instruction tuning of large language models for optimization code generation. *IEEE Transactions on Evolutionary Computation*, 30(4):1390–1404, 2026. doi: 10.1109/TEVC.2026.3656374.

Bertrand Meyer. Applying 'design by contract'. *Computer*, 25(10):40–51, 1992. doi: 10.1109/2.161279.

Sumeet Ramesh Motwani, Chuan Du, Aleksander Petrov, Christopher Davis, Philip Torr, Antonio Papania-Davis, and Weishi Yan. AutoOR: Scalably post-training LLMs to autoformulate operations research problems. In *Conference on Language Modeling (COLM)*, 2026.

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. *arXiv preprint arXiv:2603.25158*, 2026.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, pp. 9695–9717, 2024.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025. doi: 10.1007/s11704-024-40678-2.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2022.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997.

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. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.407.

Stefan Szeider. CP-Agent: Agentic constraint programming. In *Proceedings of the 3rd International Workshop on Large Language Models For Code*, pp. 6–13. ACM, 2026. doi: 10.1145/3786181.3788711.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024. doi: 10.1007/s11704-024-40231-1.

Qihao Wang, Ziming Cheng, Shuo Zhang, Fan Liu, Rui Xu, Heng Lian, Kunyi Wang, Xiaoming Yu, Jianghao Yin, Sen Hu, Yue Hu, Shaolei Zhang, Yanbing Liu, Ronghao Chen, and Huacan Wang. MemGovern: Enhancing code agents through learning from governed human experiences. *arXiv preprint arXiv:2601.06789*, 2026.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan 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*, pp. 48519–48537, 2024.

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. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026a.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 23826–23854. Curran Associates, Inc., 2023. doi: 10.52202/075280-1035.

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. *arXiv preprint arXiv:2605.23904*, 2026b.

Zhaochen Yu, Yingcheng Wu, Zhenfei Yin, Kaiyuan Chen, Zhe Zhao, Mengdi Wang, Shuicheng Yan, and Ling Yang. Recursive experiential-working memory evolution for long-horizon agent harnesses. *arXiv preprint arXiv:2608.24876*, 2026.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y. Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, pp. 86069–86100, 2026.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Yiwen Hu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Frontiers of Computer Science*, 20(12):2012627, 2026. doi: 10.1007/s11704-026-60308-3.

Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, Xue Wang, Yue Zhuan, Wang-Cheng Kang, Renkai Xiang, Heng Huang, Xinwu Cheng, and Yunsong Guo. Dream-RSI: Recursive self-improvement through evolving worlds. *arXiv preprint arXiv:2609.14858*, 2026a.

Yining Zheng, Haiyang Wei, Jiahao Lu, Linqi Yin, Yunke Zhang, Chengguo Xu, Hetao Cui, Tianxiang Sun, Shuang Chen, and Xipeng Qiu. Investigating effective LLM-based in-context tool use: What matters and how to improve. *Frontiers of Computer Science*, 20(7):2007323, 2026b. doi: 10.1007/s11704-025-41365-6.