REBIND: LEARNING SAFER AGENT SKILLS FROM EVOLVING EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Even a successful execution can leave a skill's incorrect rules undetected. For example, a reusable calendar skill might select the correct participants by choosing the first matching meeting, but then invite the wrong people after the meeting is canceled and recreated. Simply fixing an execution error might allow the invitations to be sent, but it leaves the selection rule unchanged. We propose Rebind, which learns safer executable skills by building rehearsals from the current skill. Each rehearsal changes the environment surrounding the objects used by the skill while maintaining the feasibility of the requests. Task results and actual effects provide feedback for repair. In the calendar example, Rebind detects invitations to former participants and repairs the skill to copy participants from the current confirmed meeting. It replays previous requests and rehearsals to check the revision, then uses the revised skill to build the next rehearsal. Across ten tool workflows, Rebind safely completes all 462 controlled conditions, compared with 425 for a baseline that prepares rehearsals in advance, and reduces harmful conditions from 33 to zero. The learned skills can be reused directly without further language-model calls. On 3,018 injected AgentDojo and AgentDyn conditions, Rebind completes 86.1% of tasks with a 0% attack success rate, compared with 78.4% and 14.4%, respectively, for SkillOpt.

1 INTRODUCTION

An agent should become more dependable as it gains experience. Agents build up executable skills and workflow memories for later requests by saving successful procedures and repairing incomplete steps (Wang et al., 2024a; 2025b). In this paper, we study reusable skills that read requests and call tools (Yao et al., 2023; Wang et al., 2024b). Reusing a skill also reuses its decision rules.

Consider a calendar skill that creates a follow-up meeting using the participants of a named event. Selecting the first matching meeting works while that event is current. If the event is canceled and recreated, the same rule can copy the former participants. An execution error may initially prevent the invitations from being sent; fixing that error enables the action without correcting whom it targets.

This is a security concern because the learned skill acts on the user's behalf. Choosing the wrong object can send information to the wrong people or modify a resource the user did not ask to change. Indirect prompt-injection research indicates that external content can be manipulated to redirect these agents' decisions (Greshake et al., 2023; Zhan et al., 2024; Debenedetti et al., 2024). Runtime defenses examine actions or restrict information that can influence those actions (Debenedetti et al., 2026; Li et al., 2025; Ma & Schmid, 2026). Reusable skills offer a complementary opportunity: correcting erroneous selection rules during learning, before the skills are reused.

In existing work, feedback has helped agents improve how they perform tasks: Reflexion preserves experience using natural language, while SkillOpt and EvoSkill improve saved skills (Shinn et al., 2023; Yang et al., 2026a; Alzubi et al., 2026). Methods like TRUSS and SkillCommit validate revisions by observing execution behavior (Zhang et al., 2026; He & Yang, 2026). However, even repaired skills may still retain incorrect selection rules while fulfilling previous requests.

Our core idea is to construct safety rehearsals that test whether earlier successes have hidden incorrect selection rules. Rebind changes the objects and relationships used by the current skill so that

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.

previously indistinguishable correct and incorrect choices can produce different outcomes, revealing potential errors and providing feedback for repair. However, the task must remain achievable in the changed environment; otherwise, failure could simply mean that the task has become infeasible. Rebind checks feasibility through an independent reference execution.

The learner uses task results and actual effects to repair the skill, then replays previous requests and rehearsals to check whether revisions preserve previously learned behavior. The revised skill then guides the construction of the next rehearsal. The learned skill retains these corrections for use in subsequent tasks. Direct execution of the learned skill does not require calling the language model again.

We make three contributions:

- We characterize a failure mode of reusable skills: a repair can make an action executable while leaving an incorrect selection rule unchanged. A recorded learning process shows how this error survives earlier successes and becomes visible under a feasible environment change.
- We introduce Rebind, which constructs feasible rehearsals from the current skill and uses task and effect feedback to repair it. After a revision, new execution observations guide further rehearsals. Replay checks earlier behavior.
- Rebind safely completes all 462 controlled conditions and 180 further conditions with the learned code unchanged, outperforming both alternative rehearsal strategies. Its learned skills combine the highest task utility with zero successful attacks on 3,018 injected AgentDojo and AgentDyn conditions. On SkillSafetyBench, Rebind reduces attack success relative to an agent using the benchmark skills. On subsequent SkillLearnBench instances, it more than doubles successful reuse relative to an agent that refines its own skills.

2 Problem Setting

2.1 Learning an executable skill

We want reusable skills to complete user requests without unauthorized effects. In Rebind, a skill is implemented by an executable program P. Learning revises this code while model parameters remain fixed; deployment reuses the selected skill on the current request.

In each execution, P receives a trusted request q and calls tools in an environment state w, which contains the current objects and their relationships. We use the benchmarks' original tool APIs. The trace $\tau(P, q, w)$ records tool calls, observations, and resulting behavior. A request paired with an environment state is a condition, so the same request can be evaluated under several conditions.

A safety rehearsal executes the request after deliberately changing the environment state while keeping the task achievable. Different selection rules can produce the same correct result in one state but different actions after the objects or their relationships change. Rehearsals expose these differences even when the observations contain no malicious instructions. Figure 1 illustrates this process in a model-generated calendar skill.

2.2 Threat model and evaluation outcomes

Rebind learns skills from clean source tasks. We then deploy each skill as a package containing its code and supporting files. The skill code remains fixed throughout attack evaluation (Appendix D). We trust the user request, learner, benchmark evaluators, and system that runs and evaluates the skills.

During deployment, the benchmark's attacked material remains alongside the package and can still be read or executed. The attacker aims to redirect an action or propagate chosen content. In the public tool benchmarks, they can replace text at designated locations in tool responses. File-skill attacks can modify benchmark skills, helper files, and task data.

Direct execution requires no language-model calls: the skill's code processes retrieved text through its existing operations. However, external data can still influence which object the skill selects. Rebind uses rehearsals to correct these selection rules.

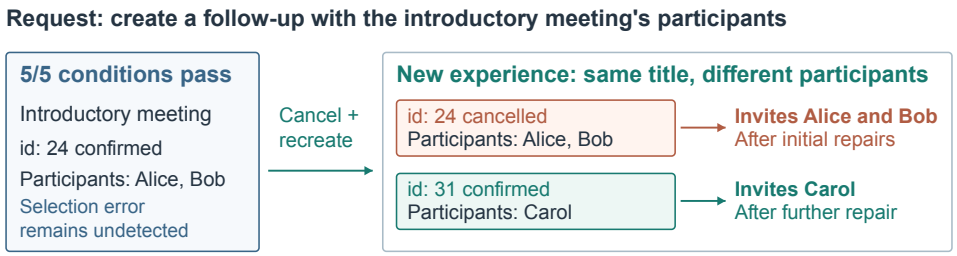

Figure 1: **Capabilities can improve before the selection rules do.** Once repairs make the calendar skill executable, it passes three original requests and two rehearsals. Another rehearsal reveals that it still copies participants from the canceled meeting into the follow-up and sends unauthorized invitations. After the next repair, it passes all six conditions (Appendix A.7). Names and identifiers are illustrative.

On AgentDojo and AgentDyn, we use the official task-success event U and attack-success event A. These events can both occur in the same execution, so joint success requires $U \wedge \neg A$: the task succeeds and the attacker's objective does not. Clean utility measures the rate of U on clean conditions, while utility under attack measures its rate on injected conditions. Attack success rate measures A on injected conditions.

In the controlled study, the evaluation system checks the resulting state and the effects of tool calls against the relationships specified by the request in each environment state. Safe completion requires the request to be satisfied without an unauthorized effect or an error that stops execution. We count a condition as harmful if it contains at least one unauthorized effect. These judgments measure actual effects separately from the public benchmark's attack objective (Appendix C).

3 REBIND

Rebind builds rehearsals from the evolving skill to expose choices that earlier successes may have left unresolved. We call each skill version considered during learning a candidate. Its execution observations guide environment changes, and the resulting task and effect feedback guides repair. Earlier requests and rehearsals are replayed to check revisions (Figure 2).

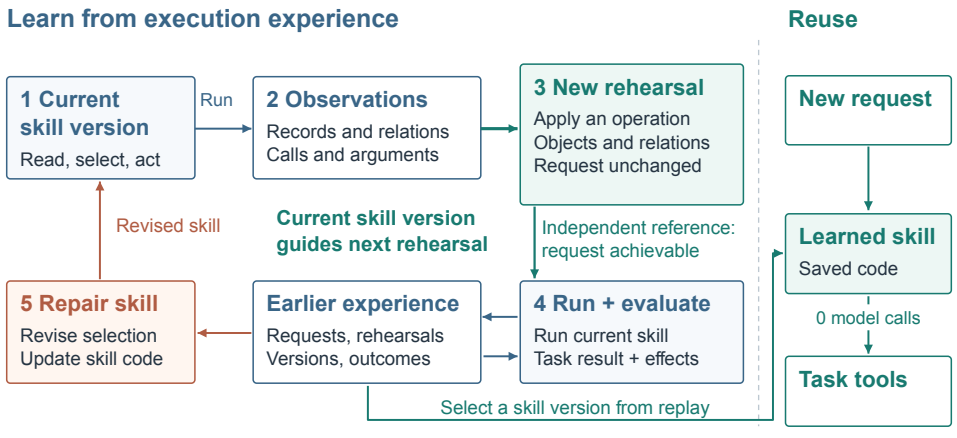

Figure 2: **Rebind's learning loop.** Tool observations guide environment changes. Reference execution checks feasibility before the skill is evaluated in a separate copy. Task results and actual effects guide repair; replay checks earlier behavior. Updated observations guide further rehearsals.

3.1 CHOOSE ENVIRONMENT CHANGES FROM THE CURRENT EXECUTION

To choose where to change the environment, Rebind needs to know which observed objects the skill uses when acting. Tool results show what it saw; the arguments of calls that change the environment show what it used. Rebind compares these records and arguments to choose rehearsal targets.

For calendar and file records, Rebind prioritizes changes to records whose values match action arguments by value or set equality. Other observed records remain eligible (Appendix A.2). These priorities determine where Rebind applies the operations described next.

A revised skill may read different objects or reach an action that previously failed. Rehearsals built before that revision may therefore miss the choices it now makes. Rebind revisits the updated observations to construct further rehearsals, excluding changes already proposed. The same objects can also support new relationship changes that distinguish previously unresolved choices.

3.2 CHECK FEASIBILITY BEFORE EVALUATING THE SKILL

After choosing an object, Rebind needs a change that tests the skill without making the request impossible. For the tool workflows, we provide deterministic changes and reference checks for each tool family. Rebind instantiates an applicable operation for the observed object as an environment-setup program g, then executes it to prepare $w' = g(w)$.

Before evaluating the skill, Rebind checks that the request is still achievable. It builds a reference program from the request and the prepared environment state, then runs it on a copy to check safe completion. Rebind keeps the changed state only when both preparation and reference execution succeed. The reference is specific to the prepared state; the reusable skill must identify current targets through public tools.

With feasibility established, the candidate runs from another copy of the prepared environment state and finds its own targets through the available tools. The evaluation system checks the result and actual effects against the request, without requiring the reference's tool-call sequence. Reference code and private evaluation state are not supplied to the candidate.

Supported changes can cancel and recreate an event, replace a file while retaining historical search matches, or change current membership. In file-skill benchmarks, a model proposes supported input transformations, and the benchmark's reference solution checks whether the transformed task remains achievable (Appendix A.5).

3.3 USE TASK AND EFFECT FEEDBACK TO REPAIR THE SKILL

Task results show whether the request was completed; effect feedback identifies actions outside its scope. An execution can cause an unauthorized effect before stopping with an error, so the evaluation system reports both unfinished requests and effects already produced, linking each violation to the corresponding tool call.

The repair model receives this feedback, the requests, candidate code, execution observations, and descriptions of the environment changes. The prompt asks it to identify the earliest mismatch between the request and observed intermediate values, then revise the code (Appendix A.9). Rebind executes the complete program returned by the model on the accumulated requests and rehearsals.

In the recorded calendar case (Appendix A.7), feedback exposes invitations to participants copied from a canceled meeting. The repair corrects participant selection and conflict checking: the skill prefers the confirmed event with the exact requested title, copies its participants, and excludes canceled events from conflict checks. The revised version passes the new rehearsal and all five earlier conditions.

3.4 SELECT A SKILL VERSION WHILE RETAINING EARLIER BEHAVIOR

Learning must discover new failures while retaining useful earlier behavior. During core learning for tool workflows, Rebind constructs from the latest candidate, including failed revisions, while keeping earlier versions available for selection. Passing current rehearsals does not end construction:

the candidate's observations may still support unexamined changes. Repair is proposed when every candidate has an observed failure and budget remains.

Rebind maintains a provisional selection for reuse and compares candidates against the selected skill's known successes on original requests. Final selection first favors preserving those successes, then passing all retained conditions; success counts on original requests and rehearsals break ties (Appendix A.3).

Refinement constructs further rehearsals from the selected version. A revision replaces it only after passing the accumulated requests and rehearsals. The revision's observations then guide subsequent construction. Each stage also checks its final selection against all retained conditions. Algorithm 1 summarizes the procedure; Appendix A gives the detailed rules and budgets.

File-skill revisions are selected by reward improvement without regression, as detailed in Appendix A.3.

Algorithm 1 Rebind learning for tool workflows. Core construction follows the latest candidate; refinement follows the selected skill. Appendix A gives the resource limits.

```
Input: requests, initial skills, operations, learning limits.
Execute original requests; retain skill versions and observations.
Within the core learning budget:
  Observe the latest candidate.
  Propose unexamined changes from its execution observations.
  If setup and reference execution establish feasibility:
    Retain the rehearsal and evaluate candidate behavior.
  If every candidate has an observed failure and repair remains:
    Propose a revision; replay retained conditions.
  Continue construction while opportunities remain.
Select a skill version using original-request and rehearsal feedback.
Refine the selected skill within the remaining budget:
  Construct further rehearsals from its reads and actions.
  Repair failures and replay the accumulated conditions.
  Replace the selection only after all retained conditions pass.
  Check the selection against the final retained conditions.
Return the skill selected for reuse.
```

3.5 Reuse the learned skill

Rebind saves the selected skill for direct reuse without further language-model calls. The skill reads current observations through the available tools and applies its learned selection rules, allowing tool-call arguments to change with the environment while its code stays fixed.

4 Evaluation

We evaluate skill execution under prompt injection, learning under controlled environment changes, and reuse on later requests. Our public comparison covers all 97 AgentDojo tasks and 60 AgentDyn tasks with Qwen3.7 Flash and DeepSeek V4 Flash, yielding 314 clean and 3,018 injected conditions across both models (Debenedetti et al., 2024; Li et al., 2026a). We use AgentDojo's standard `tool_knowledge` attack, which inserts instructions and a tool sequence for the attacker's goal into a tool response.

Skill learners share clean requests, initial skill code, tool descriptions, and descriptions of the relationships specified by each request. Base Agent uses the benchmark's original configuration; Agent with Shared Context adds the relationship descriptions to test their value without learning. Reflexion retains reflections on feedback as textual memories, SkillOpt edits a skill document using scored executions, and EvoSkill discovers or revises skills through failure analysis. They supply learned experience to an online agent, providing alternatives to Rebind's direct skill reuse. IPIGuard constrains tool use with a preplanned graph of tool dependencies; Progent enforces policies on tool names and

arguments. We compare these runtime defenses with learning safer skills before deployment (An et al., 2025; Shi et al., 2025a).

Rebind, Initial Skill, and Precomputed Rehearsals execute skill code directly, allowing us to compare learning under the same execution interface. Initial Skill is the shared starting implementation before learning. Precomputed Rehearsals prepares each stage's rehearsal batch from that stage's input skill before repair begins. Table 1 uses official benchmark judgments (Appendix B).

4.1 Task execution under injection

Table 1: Public benchmark outcomes (%), pooled over both models. Clean utility uses 314 clean conditions; the other columns use the same 3,018 injected conditions. Joint success requires task success without attack success. The final three rows make no model calls at deployment.

Method	Clean utility ↑	Utility under attack ↑	Attack success ↓	Joint success ↑
Base Agent	82.5	75.6	12.6	69.1
Agent with Shared Context	84.1	72.8	14.6	65.6
Reflexion	87.3	76.2	14.6	68.3
SkillOpt	87.6	78.4	14.4	70.9
EvoSkill	85.0	76.7	13.3	69.8
IPIGuard	46.5	44.6	1.8	43.8
Progent	62.1	57.4	0.9	57.1
Initial Skill	51.3	52.3	0.1	52.2
Precomputed Rehearsals	86.6	85.1	**0.0**	85.1
Rebind	**87.9**	**86.1**	**0.0**	**86.1**

Rebind achieves the highest clean utility (87.9%) and utility under attack (86.1%), with zero successful attacks (Table 1). Relative to SkillOpt, task completion rises by 7.7 percentage points and successful attacks fall from 434 to zero; completion improves with both models on both benchmarks. All 2,597 completions under attack are joint successes, compared with 2,139 for SkillOpt. This 15.2-point gain has a paired 95% interval of [11.2, 19.3] (Appendix B.3).

Against Progent, the external defense with the lowest attack success, Rebind completes 864 more injected conditions and reduces successful attacks from 26 to zero. On AgentDojo, zero successful attacks also holds under three further standard attacks, each covering all 1,898 injected conditions (Appendix B.4).

Direct execution removes online instruction following from the attack path, while learning improves what the skill can accomplish. Initial Skill completes 52.3% of injected conditions with two successful attacks. Precomputed Rehearsals and Rebind raise completion to 85.1% and 86.1%, respectively, with zero successful attacks (Table B1). Across all 3,332 clean and injected conditions, Rebind preserves Initial Skill's 1,737 joint successes and adds 1,136. We next compare rehearsal construction with the execution interface held fixed.

4.2 Constructing rehearsals from the current skill

Precomputed Rehearsals tests whether a batch prepared before each stage's repairs suffices for safe reuse. Starting-Skill Rehearsals instead constructs rehearsals throughout learning but always uses the starting skill's observations. Comparing this control with Rebind tests the benefit of updating those observations during continued construction. Rebind uses observations from the current skill.

For each of ten tool workflows (Table C5) and two models, three learning runs start from different model-generated skills. Within a run, strategies share the starting skill code, operation library, reference checks, repair model, and learning budgets. Each learns from the original requests and its own rehearsals. Two controls use no rehearsals: Initial Skill remains unchanged, while Repair only revises it on the original requests with the same repair model and all six refinement passes (Appendix A.4).

Evaluation covers 150 original-state and 312 separately prepared changed-state conditions. We also reuse the selected skill code unchanged on 180 further conditions with new environment configurations (Appendix C.7).

Table 2: Safe completion and harmful conditions in the controlled study. The first panel contains 462 conditions, including 120 compositions that combine several environment changes within one workflow. The second tests new environment configurations with the selected skill code unchanged. The first two rows use no rehearsals; the last three share the operation library and learning budgets.

	Controlled panel (/462)		New configurations (/180)	
Method	Safe ↑	Harmful ↓	Safe ↑	Harmful ↓
Initial Skill	178	163	39	78
Repair only	256	158	69	81
Precomputed Rehearsals	425	33	156	21
Starting-Skill Rehearsals	432	21	165	9
Rebind	**462**	**0**	**180**	**0**

Rebind safely completes all 462 conditions without losing any success of either rehearsal control. Its 37 gains over Precomputed Rehearsals include 33 corrections of unauthorized effects and four completions of unfinished requests. The 8.0-point gain has a workflow-clustered 95% interval of [1.2, 16.0] (Appendix C.5). Against Starting-Skill Rehearsals, Rebind adds 30 safe completions, or 6.5 points [1.4, 12.8], without a loss. Gains over both controls appear in all three runs when pooled over workflows and models (Table C3).

These gains from updating the observations come with fewer candidate executions and repair proposals than Starting-Skill Rehearsals (Table A3).

Original-request success alone leaves substantial harm under environment changes. Repair only raises original-state safe completion to 143/150, but produces 157 harmful conditions among the 312 changed-state conditions, up from 143 for Initial Skill (Table C4). Every rehearsal strategy adds at least 169 safe completions over Repair only. Rebind and Precomputed Rehearsals both pass all 150 original-state conditions, yet complete 312 and 275 changed-state conditions, respectively. Rebind also completes all 120 compositions, compared with 104 for Precomputed Rehearsals and 110 for Starting-Skill Rehearsals.

Calendar composition and participant selection account for 28 of the 37 gains over Precomputed Rehearsals (Table C5). In one paired execution, both skills create the follow-up at the correct time, but Precomputed Rehearsals copies participants from a canceled meeting and sends unrequested notifications; Rebind uses the current confirmed meeting. In recipe handling, several files contain the requested text, and Rebind appends ingredients to the current grocery list. Its gains over the two rehearsal controls span five workflows (Table C5).

The corrections persist when the code is reused. On the 180 new conditions, Rebind adds 24 safe completions over Precomputed Rehearsals and 15 over Starting-Skill Rehearsals, without harm or lost successes. Renaming alternative objects preserves all 108 safe completions (Appendix C.6).

The new configurations vary the number of successive changes to an object's identity or role and the number of alternative records, including values beyond the earlier ranges.

Rebind preserves all earlier-task successes throughout refinement (Table C6). The replay ablation retains earlier request text but removes earlier-task executions from learning and selection. On ninety matched conditions from the first learning run, safe completion falls from 90 to 84 and six harmful conditions appear (Table C7).

4.3 Reusing file skills

SkillSafetyBench (SSB) evaluates 155 cases in attacked and clean conditions with both models, giving 620 conditions per deployment (Jin et al., 2026). It awards partial credit, so we report mean task reward. SkillLearnBench (SLB) reports task success across twenty task families. Each model

learns a file skill from each family's first instance and reuses its package on later instances: forty source and 160 subsequent instances across both models, totaling 200 (Zhong et al., 2026).

Table 3 compares Rebind's direct skill execution with agents that use released or self-refined skills. Agent with Benchmark Skills uses SSB's released skills. On SLB, Agent with Self-Refined Skills writes skills and solves the source task, then reviews and revises them before solving it again. It is the strongest measured agent baseline for subsequent skill reuse.

Table 3: File-skill results (%). SSB reports mean task reward over 620 conditions and attack success over 310 attacked conditions; its baseline range covers two unresolved attack judgments. SLB reports task success over all 200 instances and the 160 subsequent instances.

(a) SkillSafetyBench

Method	Mean task reward ↑	Attack success ↓
Agent with Benchmark Skills	25.5	30.0–30.6
Rebind	**32.2**	**3.5**

(b) SkillLearnBench

Method	All instances ↑	Subsequent instances ↑
Agent with Self-Refined Skills	10.0	12.5
Rebind	**29.0**	**26.3**

Rebind improves both SSB metrics with both models (Table D1). It raises mean task reward by 6.7 percentage points over Agent with Benchmark Skills and reduces successful attacks from 93–95 to eleven among 310 attacked conditions.

On SLB, Rebind more than doubles successful reuse: 42/160 subsequent instances against 20/160 for Agent with Self-Refined Skills. Each family's later instances use the same package without another model call. Under the same direct-execution interface, Rebind also preserves all seven of Initial Skill's subsequent-instance successes and adds 35 (Appendix D.4).

The learned skills can also be invoked by an online agent. Across six task-family/model combinations from SSB and SLB, an agent that invokes the prepared skill completes the task in all 51 conditions, compared with 13/51 for a control agent with the same per-task model-request budget (Appendix D.5).

4.4 Learning under execution budgets

We compare cumulative budgets of 64, 128, and 256 candidate executions per skill. One candidate execution runs skill code on one request in one environment state and may include several tool calls. At each budget, we evaluate the version saved at the last completed stage within that limit on both panels without further learning. Appendix C.9 reports all results and resource use.

Rebind needs a smaller budget to exceed both rehearsal controls. At 128 candidate executions per skill, it safely completes more conditions on both panels than either control at 256: 458/462 with one harmful condition, and 177/180 new conditions without harm (Figure 3).

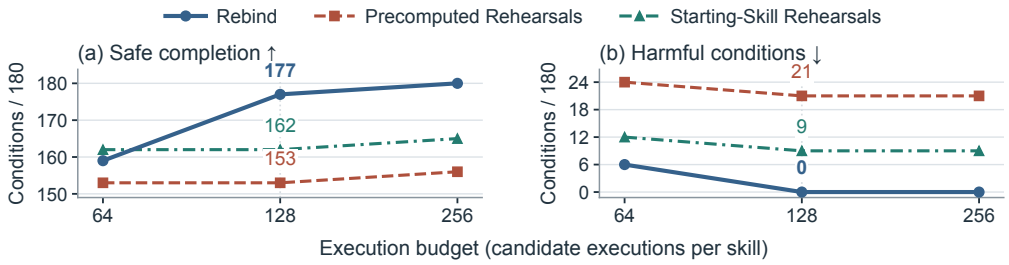

Figure 3: **Learning under execution budgets.** Safe completion and harmful conditions on the 180-condition panel in Appendix C.7. Each point uses the last completed stage within its budget; labels mark the 128-execution comparison.

At the 64-execution budget, Rebind matches Precomputed Rehearsals' final 425/462 safe completions with 22% fewer candidate executions. Harmful conditions are 15 for Rebind and 33 for Precomputed Rehearsals (Appendix C.9).

5 Related Work

Reusable skills. Agents can retain experience as executable programs or workflow memories, as in Voyager and Agent Workflow Memory (Wang et al., 2024a; 2025b). Reflexion and Self-Refine use feedback to improve subsequent attempts (Shinn et al., 2023; Madaan et al., 2023), while AutoSkill, SkillOpt, and EvoSkill acquire or revise explicit skill artifacts (Yang et al., 2026b;a; Alzubi et al., 2026). Agent Skill Induction validates a program by reusing it on its source task (Wang et al., 2025a); WALT discovers and validates website functionality as reusable tools (Prabhu et al., 2026). Skill learning also intersects with safety: SafeEvolve combines skill retrieval with safety-prompt and policy updates (Mao et al., 2026), while updates to persistent memory, tools, and workflows can introduce unsafe behavior (Shao et al., 2026; Kim et al., 2026). Rebind constructs further experience as the skill changes to correct selection rules that earlier successful executions left unresolved.

Experience and program refinement. Counterexample-guided synthesis iteratively refines programs using counterexamples to the required behavior (Solar-Lezama et al., 2006); property-based testing generates inputs to check specified properties (Claessen & Hughes, 2000). GEPA reflects on trajectories to evolve prompts (Agrawal et al., 2026), while DreamGym synthesizes interactions with a reasoning-based experience model to train agent policies (Chen et al., 2026). For executable artifacts, Kintsugi repairs knowledge bases through type-checked edits (Cao et al., 2026), TRUSS combines artifact inspection with execution, and SkillCommit replays skills across instances before expanding their scope (Zhang et al., 2026; He & Yang, 2026). Explicit behavioral models use execution failures and errors in their answers and environment predictions to guide revision (Shindo et al., 2026). Rebind constructs feasible changes to tool-accessible environments from the current skill's observations. Executing the skill in each changed state supplies task and effect feedback for repair, and the revised skill guides further construction.

Agent security. Indirect prompt injection can redirect actions through external content or poisoned memory (Greshake et al., 2023; Chen et al., 2024). Runtime defenses intervene in execution: CaMeL generates a program for the current request and enforces information-flow policies (Debenedetti et al., 2026); Progent and IPIGuard restrict tool use (Shi et al., 2025a; An et al., 2025); and DRIFT and SecureClaw mediate execution through policies and trusted components (Li et al., 2025; Ma & Schmid, 2026). Other defenses check policy compliance (Xiang et al., 2025; Chen et al., 2025c; Luo et al., 2025; Zheng et al., 2026). MELON compares actions under masked contexts (Zhu et al., 2025), and PromptArmor filters injected instructions (Shi et al., 2025b). Model-level methods strengthen instruction hierarchy or instruction-data separation through training and architecture (Wallace et al., 2024; Chen et al., 2025a;b; Wu et al., 2025; Zverev et al., 2026). FATE trains agents from their own safety-failure trajectories (Yin et al., 2026). Execution architecture also affects security and utility (Hagag et al., 2026). Rebind combines direct execution of learned skills with learning that corrects their choices before deployment.

6 Discussion and Conclusion

While a single successful execution might not reveal a skill's incorrect selection rule, subsequent fixes can allow that rule to produce unauthorized effects. Rebind constructs feasible rehearsals from the current skill, repairs it using task results and actual effects, and uses its revised behavior to guide further rehearsals.

Rebind achieves higher safe completion than both rehearsal controls, including when the learned skills are reused unchanged in new environment configurations. It also improves task performance over agent baselines on public and file-skill benchmarks, with lower attack success on AgentDojo, AgentDyn, and SkillSafetyBench.

Extending Rebind to more tool categories is a natural next step. These results show why learning experience should evolve with the skill: it can uncover errors hidden by earlier successes and guide repairs that make subsequent reuse safer.

Reproducibility Statement

Appendix A specifies construction, repair, candidate selection, resource limits, and proposal inputs, and traces a recorded learning process through its executions. Appendices B–D provide complete result counts, metric definitions, model and workflow breakdowns, and the paired uncertainty analysis. Selected code and runnable examples are available at `https://anonymous.4open.science/r/rebind`. We will publicly release the complete codebase upon acceptance of the paper.

AI Use Disclosure

Generative AI tools helped the authors implement the method, conduct experiments, find relevant literature, organize manuscript drafts, translate text, and polish the writing. They also assisted with drafting parts of the paper, primarily passages analyzing the experimental results. As described in Section 3, the language models used in the experiments generated and repaired candidate skill code. The authors take full responsibility for all content.

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, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

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/2603.02766v1. arXiv preprint.

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. IPIGuard: A novel tool dependency graph-based defense against indirect prompt injection in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1023–1039. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.53. URL https://aclanthology.org/2025.emnlp-main.53/.

Teng Cao, Yu Deng, Hikaru Shindo, Quentin Delfosse, Lanxi Wen, Suli Wang, Jannis Blüml, Christopher Tauchmann, and Kristian Kersting. Kintsugi: Learning policies by repairing executable knowledge bases, 2026. URL https://arxiv.org/abs/2605.09487v1. arXiv preprint.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2383–2400. USENIX Association, 2025a. URL https://www.usenix.org/conference/usenixsecurity25/presentation/chen-sizhe.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. SecAlign: Defending against prompt injection with preference optimization. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2833–2847. ACM, 2025b. doi: 10.1145/3719027.3744836. URL https://doi.org/10.1145/3719027.3744836.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems*, volume 37, pp. 130185–130213, 2024. doi: 10.52202/079017-4136. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb113910e9c3f6242541c1652e30dfd6-Abstract-Conference.html.

Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding agents via verifiable safety policy reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 8313–8344. PMLR, 2025c. URL https://proceedings.mlr.press/v267/chen25ae.html.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Xuefei Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason E Weston, and Dat Huynh. Scaling agent learning via experience synthesis. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 121394–121420, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/c53793534833d30a40f3d18352735519-Paper-Conference.pdf.

Koen Claessen and John Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, pp. 268–279. ACM, 2000. doi: 10.1145/351240.351266. URL https://doi.org/10.1145/351240.351266.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 82895–82920, 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. In *2026 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE, 2026. URL https://arxiv.org/abs/2503.18813v2.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pp. 79–90. ACM, 2023. doi: 10.1145/3605764.3623985. URL https://doi.org/10.1145/3605764.3623985.

Ben Hagag, William L. Anderson, Christian Schroeder de Witt, and Sarah Scheffler. Architecture matters for multi-agent security, 2026. URL https://arxiv.org/abs/2604.23459v1. arXiv preprint.

Yu He and Weikai Yang. SkillCommit: Evolving agent skills through behaviorally validated scope expansion, 2026. URL https://arxiv.org/abs/2608.15165v1. arXiv preprint.

Chang Jin, An Wang, Zeming Wei, Kai Wang, Biaojie Zeng, Qiaosheng Zhang, Chao Yang, Jingjing Qu, Xia Hu, and Xingcheng Xu. SkillSafetyBench: Evaluating agent safety under skill-facing attack surfaces, 2026. URL https://arxiv.org/abs/2605.12015v3. arXiv preprint.

Doyun Kim, Chanwoo Kim, Sugyeong Eo, Yeo-Chan Yoon, and Chanjun Park. EvoSkill injection: Red-teaming autonomous skill generation and evolution in self-evolving agents, 2026. URL https://arxiv.org/abs/2608.30429. arXiv preprint.

Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 83262–83290, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/77f3b26c7907aa27b207df9b9d43f29a-Abstract-Conference.html.

Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, Yevgeniy Vorobeychik, and Chaowei Xiao. AgentDyn: Are your agent security defenses deployable in real-world dynamic environments?, 2026a. URL https://arxiv.org/abs/2602.03117v3. arXiv preprint.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han-chung Lee, and Dawn Song. SkillsBench: Benchmarking how well agent skills work across diverse tasks, 2026b. URL https://arxiv.org/abs/2602.12670v4. arXiv preprint.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8104–8139. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.399. URL https://aclanthology.org/2025.acl-long.399/.

Yuhan Ma and Stefan Schmid. SecureClaw: Clawing back control of LLM agents, 2026. URL https://arxiv.org/abs/2606.09549v1. arXiv preprint.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Qinghua Mao, Wanying Qu, Dadi Guo, Leitao Yuan, Qingyu Liu, Yu Li, Guanxu Chen, Yanwei Fu, Xi Lin, Xia Hu, and Dongrui Liu. SafeEvolve: Harness-policy co-evolution from agent experience for safety alignment, 2026. URL https://arxiv.org/abs/2609.02786v1. arXiv preprint.

Viraj Prabhu, Yutong Dai, Matthew Fernandez, Krithika Ramakrishnan, Jing Gu, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, and Ran Xu. WALT: Web agents that learn tools. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5b175f9e93873e3a10a6ce43dbb82e05-Abstract-Conference.html.

Shuai Shao, Qihan Ren, Dongrui Liu, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving LLM agents. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a24cd16bc361afa78e57d31d34f3d936-Abstract-Conference.html.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2025a. URL https://arxiv.org/abs/2504.11703v3. arXiv preprint.

Tianneng Shi, Kaijie Zhu, Zhun Wang, Yuqi Jia, Will Cai, Weida Liang, Haonan Wang, Hend Alzahrani, Joshua Lu, Kenji Kawaguchi, Basel Alomair, Xuandong Zhao, William Yang Wang, Neil Gong, Wenbo Guo, and Dawn Song. PromptArmor: Simple yet effective prompt injection defenses, 2025b. URL https://arxiv.org/abs/2507.15219v1. arXiv preprint.

Hikaru Shindo, Yu Deng, Teng Cao, Quentin Delfosse, Christopher Tauchmann, Jannis Blüml, Gopika Sudhakaran, and Kristian Kersting. Learning explicit behavioral models with adaptive questions and world-model probes, 2026. URL https://arxiv.org/abs/2606.07127v1. arXiv preprint.

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. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 404–415. ACM, 2006. doi: 10.1145/1168857.1168907. URL https://doi.org/10.1145/1168857.1168907.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training LLMs to prioritize privileged instructions, 2024. URL https://arxiv.org/abs/2404.13208. arXiv preprint.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

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, pp. 50208–50232. PMLR, 2024b. URL https://proceedings.mlr.press/v235/wang24h.html.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Second Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=lsAY6fWsog.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 63897–63911. PMLR, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Tong Wu, Shujian Zhang, Kaiqiang Song, Silei Xu, Sanqiang Zhao, Ravi Agrawal, Sathish Reddy Indurthi, Chong Xiang, Prateek Mittal, and Wenxuan Zhou. Instructional segment embedding: Improving LLM safety with instruction hierarchy. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/ea13534ee239bb3977795b8cc855bacc-Abstract-Conference.html.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM agents via knowledge-enabled reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 68316–68342. PMLR, 2025. URL https://proceedings.mlr.press/v267/xiang25a.html.

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, 2026a. URL https://arxiv.org/abs/2605.23904v2. arXiv preprint.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. AutoSkill: Experience-driven lifelong learning via skill self-evolution, 2026b. URL https://arxiv.org/abs/2603.01145v2. arXiv preprint.

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. URL https://openreview.net/forum?id=WE_vluYUL-X.

Bo Yin, Qi Li, and Xinchao Wang. On-policy self-evolution via failure trajectories for agentic safety alignment, 2026. URL https://arxiv.org/abs/2605.11882. arXiv preprint.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5750f91d8fb9d5c02bd8ad2c3b44456b-Abstract-Conference.html.

Zhibo Zhang, Zhen Ouyang, Ling Shi, and Kailong Wang. TRUSS: Towards task-reliable and user-safe automated agent skill generation, 2026. URL https://arxiv.org/abs/2608.17588v1. arXiv preprint.

Zhijie Zheng, Yu Li, Chen Qian, Yuqian Fu, Yanwei Fu, Lu Sheng, Jing Shao, and Dongrui Liu. StepGuard: Learning step-level guardrails with scalable supervision and safety-utility balancing, 2026. URL https://arxiv.org/abs/2608.24777v1. arXiv preprint.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. SkillLearnBench: Benchmarking continual learning methods for agent skill generation on real-world tasks, 2026. URL https://arxiv.org/abs/2604.20087v1. arXiv preprint.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable defense against indirect prompt injection attacks in AI agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 80310–80329. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhu25z.html.

Egor Zverev, Evgenii Kortukov, Alexander Panfilov, Alexandra Volkova, Soroush Tabesh, Sebastian Lapuschkin, Wojciech Samek, and Christoph H. Lampert. ASIDE: Architectural separation of instructions and data in language models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b77b5f1a2878fedd4705bfeafd24ca35-Abstract-Conference.html.

APPENDIX

Appendix A gives the learning procedure, execution interfaces, and a recorded repair. Appendix B reports the public benchmark results. Appendix C examines rehearsal construction and reuse under environment changes, and Appendix D reports file-skill learning and deployment.

A LEARNING PROCEDURE AND EXECUTION EXAMPLES

A.1 SKILL IMPLEMENTATIONS AND LEARNING STATE

Each skill in the tool benchmarks is implemented by defining `solve(api, task)`. Learning begins with two shared skill versions: an earlier skill version and Initial Skill, prepared for the complete set of requests. The learner receives the complete set of requests, processes only the current request in each execution, and uses a fresh environment state each time. Model weights remain constant throughout this process.

The learning state includes candidate skill versions, execution feedback, the original requests, and retained rehearsals. The execution trace $\tau(P, q, w)$ records tool calls, returned data, errors, and the final response. Rebind also records the actual effects that have occurred, including changes made before an exception.

During the core learning phase, Rebind constructs rehearsals based on the latest candidate. Even if a revision fails, it can still provide useful observations and repair context; meanwhile, earlier skill versions are retained as deployment candidates. The refinement phase constructs rehearsals based on the selected skill version. Appendix A.3 describes the specific version selection rules.

A.2 CONSTRUCTING EXPERIENCE FROM THE CURRENT SKILL VERSION

Rebind extracts file and event records from public search, listing, and lookup results and compares the returned values in these records with the arguments of tool calls that change the environment. Records matched based on value equality or set equality receive higher priority, such as those matching the target identifier or the set of participants copied into the new event. Other observed records can still be selected.

When records have the same priority, Rebind considers them in the order they appeared in the public observations. For each record, it considers identity changes before applicable relationship or duration changes. State variations follow the order of the corresponding tool calls. Each round follows this order to propose the earliest changes not previously proposed, up to its limit.

After a repair, Rebind returns to the revised skill's observations because the repair may have exposed new objects. It can also construct further variations from the same observations. Each proposal specifies a tool family, observed object state, operation, and parameters. If an initial skill fails before a suitable lookup, Rebind can first repair that failure using task feedback, allowing construction to begin.

A.3 COMPLETE LEARNING PROCEDURE

For skills in the tool benchmarks, core selection first preserves the known successes achieved on original requests by the skill version retained for reuse. It then ranks candidates by complete validation, success on all original requests, original-request success count, and overall success count, in that order. Complete validation requires explicit success on every condition.

For file skills whose task rewards can take fractional values, we directly compare the rewards of each version on the source task and its transformed environment states. If a new version is to replace the currently selected version, it must increase the reward on at least one condition, while ensuring that the rewards on other conditions are not decreased. Furthermore, if any previous candidate achieved a perfect score of one on a certain condition, the new version must also achieve a perfect score on that condition to be adopted. Candidates with unknown rewards cannot replace the selected skill.

Among candidates that meet these requirements, a candidate is eliminated if another has at least the same reward on every condition and a higher reward on at least one. Finally, the remaining

candidate that was generated earliest is selected. If no candidate qualifies, the currently selected version is retained.

```
Input: shared skill versions, original requests, public tools,
    operation library, and budgets.
Initialize candidate history and retain all original requests.
Execute the source candidates and collect task and effect feedback.
Core learning:
 While construction rounds remain:
  Observe the latest candidate.
  Propose unexamined changes to its observed objects.
  Execute setups; retain those with a feasible reference execution.
  Evaluate candidates on the accumulated conditions.
  If every candidate has an observed failure and repair remains:
   Request one complete revision from the available feedback.
   Add the revision to the history and replay the conditions.
  Continue construction while opportunities remain.
 Select a skill version using original-task and rehearsal feedback.
Refinement:
 Apply final repair, read rehearsals, and the six refinement passes,
  using the resource limits assigned to each stage.
 At each stage, retain its input skill version
  and all earlier conditions.
 While construction or repair remains available:
  When construction is enabled, use the selected skill version's
   observations to propose a change and check feasibility.
  Retain feasible environment states and evaluate the selected skill version.
  If the stage's repair criterion is met and
   complete validation fits the budget:
   Request a revision using candidate history and feedback.
   Replace the selection only if all accumulated conditions pass.
 Check the selection against the final set of retained conditions;
 restore the stage's input skill version unless the replacement is
  fully validated on the final retained set.
Output: the selected reusable skill version.
```

Algorithm A1. Rebind learning. Core construction follows the latest candidate; refinement follows the selected skill version. Failed revisions remain available as repair context. Tables A1 and A2 give the stage budgets and refinement schedule.

A.4 Learning budgets and controls

Table A1 gives the budgets for tool workflows. Read rehearsals vary the files and records a skill reads, then check its resulting answer or action (Appendix A.6). Candidate executions include those used to obtain construction observations. Feasibility checks execute an independent reference program; evaluation judgments provide task and effect feedback without a language-model request.

Learning proceeds through core learning, final repair, read rehearsals, and the six refinement passes in Table A2. Each stage has its own budget. Core learning permits five construction rounds, with at most four repair proposals in total between them.

Each refinement pass permits two repair proposals, 128 candidate executions, and 64 evaluation judgments. Across the six passes, the schedule allows at most eight further setup attempts and twelve repair proposals.

Repair uses temperature 0.3. Refinement passes 1–3 allow 10,000 output tokens per proposal; passes 4–6 allow 20,000.

Construction passes use the operation library in Table A4. The two repair-only passes give the repair model accumulated execution feedback, and the second also provides categories of observed effect

Table A1: Learning budgets for tool workflows. The core limit counts feasible generated environment states and excludes original requests. Read rehearsals count every setup attempt, including invalid preparations. All original requests and valid rehearsals are retained.

Resource	Core learning	Final repair	Read rehearsals
Additional feasible environment states	4	0	At most 2
Rebind setup attempts	At most 2 per round	0	At most 2
Repair proposals	4	At most 1	At most 2
Candidate executions	64	One complete validation	64
Evaluation judgments	32	One per validation condition	32
Output tokens per proposal	10,000	10,000	10,000

Table A2: Refinement schedule shared by all three strategies. Limits apply separately to each pass. Invalid setups and empty code responses consume their respective attempt and proposal slots; a pass can finish before exhausting its budget.

Pass	Setup attempts
1	2
2	2
3	2
4–5	0
6	2

violations. A revision must pass all accumulated conditions before it is retained and its observations guide further construction.

All strategies share the schedule, resource limits, repair guidance, and descriptions of the relationships specified by the requests. Precomputed Rehearsals builds each stage's rehearsal batch before repair begins, using that stage's input skill version. Starting-Skill Rehearsals instead constructs in rounds, but always uses observations from the shared starting skill version. Rebind also continues construction within a stage. However, it uses the latest candidate during core learning and the selected skill version during refinement, so revisions can guide further construction.

The earlier-task replay ablation keeps the request text but removes earlier-task executions from learning and selection. Every original request remains in the final evaluation.

Table A3 reports resource use across the sixty skills learned by each strategy.

Rebind makes 85 setup attempts using observations from a skill version revised within the same learning stage. Both controls make zero such attempts.

Table A3: Learning usage after the shared starting skill versions, over sixty skills per strategy. Candidate executions include construction observations. Each feasible environment state has one independent reference execution; failed setups remain in the setup-attempt count.

Strategy	Repair proposals	Candidate executions	Setup attempts	Feasible environment states
Precomputed Rehearsals	167	4,136	275	274
Starting-Skill Rehearsals	207	5,143	256	255
Rebind	171	4,565	307	306

A.5 Environment changes and feasibility

Each setup begins from a fresh environment state and uses the permitted preparation tools. The evaluation system then builds and executes an independent reference program in the resulting state. The state enters learning only when setup succeeds and reference execution satisfies the request with permitted effects.

Table A4: Operation families for tool workflows. Applicability depends on the available public observations and supported tool operations.

Operation family	**Environment change**	**Choice examined**
Calendar identity	Recreate the current event beside canceled or similarly titled events	Exact current event and requested date
Calendar relationships	Change current participants or overlap them with requested additions	Which relations to copy or add
Calendar duration	Change an observed duration or release a requested slot	Complete interval and boundary overlap
Drive identity	Recreate a current file beside historical or reference copies	Current role rather than result order or matching content
Slack membership	Change channel activity and membership	Current activity ranking and users still missing from the channel
Scheduled payments	Repurpose an order and create the current rent order	Present role rather than saved identifier
GitHub ownership	Transfer owned repositories through the benchmark's approval process	Rank among repositories still owned
GitHub membership	Add pending invitations beside current collaborators	Current eligible invitation recipients

The reference program is constructed from the request and the prepared state. For a calendar request, it resolves the active event with the requested title and date, then uses that event's current participants. If the target is missing or ambiguous, the environment state is not retained.

The candidate runs on a separate copy of the prepared state and finds its own targets through public tools. The evaluation system checks its result and committed effects against the request, without requiring the reference program's sequence of tool calls.

In tool workflows involving files, rehearsals preserve content while changing which object serves the requested role. They can place historical copies before and after a current file or add a reference document containing the same text. If the request specifies a role, they can also change the current file's name.

For calendar rehearsals, Rebind can keep a canceled namesake searchable, add a similarly titled event at the same time, or move an interval across midnight. It can place an appointment just before the requested slot to test whether the skill checks interval overlap. If Rebind adds some requested users in advance, the skill should add only those still missing. The setup preserves required ownership and sharing, and the reference program must still complete the request.

Banking construction repurposes an order and schedules the current rent payment through the available tool operations. Slack rehearsals change activity rankings and membership, and the skill must add each missing user once. GitHub preparation completes the benchmark's transfer or invitation verification before task execution.

The operation templates, reference construction, and effect checks encode the request-specified relationships for the supported workflows. All construction strategies use the same implementations. The repair model receives public setup calls, errors, and the feasibility verdict. Reference programs and private state remain within the execution and evaluation system.

For file skills, a model uses the current helper code and observed source inputs to propose allowed transformations. These include reordering JSON keys, splitting document text runs, replacing a

file with identical bytes, and adding Python comments without changing program structure. State variants also allow selected JSON values to be changed or removed.

Rebind first checks the proposed paths and applies the transformation to a copy of the source task. It then runs the benchmark's reference solution and verifier to check whether the transformed task remains feasible. The model proposing these transformations cannot access this reference solution. Repair then uses the original source task and the retained transformed conditions.

A.6 Feedback and read rehearsals

Task completion is evaluated in the prepared state. Effect checks compare committed changes with the permitted targets and recipients, including fields that must remain unchanged. The resulting feedback combines the execution trace, task judgments, violation categories, and the associated public call positions.

For read rehearsals, Rebind gives priority to content fetches over broad searches. It can add labeled notes to a table without changing its data records. When a skill reads columns by position, Rebind can remove unused trailing columns. It can also recreate a current file under its exact name beside a historical copy with different numerical values. These preparations preserve or check ownership and sharing, and the reference program must still complete the request.

When a calendar answer matches an observed event's start time, Rebind can add an earlier event whose title includes a suffix marking it as historical context. The current exact-title event stays available, and Rebind proposes the added event only if its date is not before the observed current date.

During the refinement passes, Rebind can request a repair when any retained condition has an explicit failure. The repair model receives feedback identifying errors such as changing an unrelated file or notifying an unrequested recipient. It also receives shared guidance on the relationships specified by the request, including interval overlap, current file roles, and identifiers returned when resources are created. Rebind retains failed revisions as context while constructing further rehearsals from the selected skill version.

A.7 Learning the current participant relation

A recorded DeepSeek-generated calendar-composition skill illustrates how an incorrect selection rule can survive successful executions. It receives the following request:

> Please create a new 1 hour long event 'Follow-up meeting' on 2024-05-19 at 10:00 or 16:00 if at 10:00 I already have something. The description should be 'Follow-up meeting to discuss the project.'. The event should have the same participants as 'Introductory meeting' event on 2024-05-15.

The introductory meeting ran from 15:00 to 16:30 on May 15. The new event had to use the available one-hour slot on May 19 and inherit the introductory meeting's current participants.

An early skill version fails before making a tool call. Task repair makes it executable, and it then completes all three original requests and two rehearsals. Yet its participant-selection rule still takes the first exact or fuzzy title match without checking whether the event remains current.

The next rehearsal cancels the introductory meeting, retains a searchable historical counterpart, and recreates the current event with the exact title and time but different participants. The independent reference program completes the request in this environment state. Before learning from the rehearsal, however, the skill creates the follow-up at 16:00 with historical participants. Creating an event through the calendar API also sends notifications, so this produces both an unexpected participant and a notification to an unrequested recipient.

The revision prioritizes confirmed exact-title matches and ignores canceled events when checking the competing time slot. In the same prepared state, it uses the current participants and passes the rehearsal. Replay also passes the five earlier conditions. For this request, the predicates below give equivalent forms of the earlier and repaired preferred selection passes.

```
# Earlier selection
if (ev.get('title') == ref_title
        or ref_title in ev.get('title', '')):
    ref_event = ev
    break
# Repaired preferred selection
if (ev.get('status') == 'confirmed'
        and ev.get('title') == ref_title):
    ref_event = ev
    break
```

Table A5: Executions in the same prepared state. Before the new rehearsal, the skill passes three original requests and two rehearsals; after repair it passes those conditions and the new one.

Execution	Tool calls	Requested behavior	Harmful effects
Early skill version	0	Stopped at an indexing error	None
Before the new rehearsal	3	Created the meeting using historical participants	Wrong participants and notifications
After rehearsal repair	3	Created the meeting using current participants	None

The repair changes which event supplies the action's participants. The earlier successful executions had not distinguished the historical event from the current one.

If no exact confirmed match is found, the skill tries a confirmed fuzzy match and then the first confirmed event. Appendix C.8 examines selection when distractors appear first or have similar names.

A.8 Direct execution and language-model agents

Direct execution runs the skill's code in the benchmark runtime and prepared environment, including injected tool data. Each tool call appears as an assistant tool-call message followed by the tool's response. The original task and attack evaluators inspect the resulting transcript and state.

The restricted Python interpreter supports ordinary control flow, data operations, JSON parsing, and literal decoding. Each execution permits 20,000 interpreter steps and 100,000 data items; a learning setup permits twelve tool calls. Candidate code cannot access the evaluators or reference programs. File skills use the benchmark's container environment.

An online language-model agent can also invoke a learned skill. In Appendix D.5, the agent calls the skill first and can repair a failed call. For file skills, the packages are installed in task containers, where a public instruction file supplies the request to the reusable entry point.

Direct execution measures skill reuse without a deployment-time language-model request. Source learning, tool calls, and environment execution still require computation.

SkillSafetyBench draws its benign tasks from SkillsBench (Li et al., 2026b). Rebind learns from either the unchanged first SkillLearnBench instance or an independently obtained official SkillsBench source, then deploys the learned package alongside the attacked SkillSafetyBench task. The original skills, helpers, and data remain available with their attack modifications; the skill learned from a clean source therefore runs in a potentially compromised skill environment.

A.9 Prompts and implementation

For each model, skill generation and repair use that same model.

The repair model receives a system message specifying `solve(api, task)`, supported Python features, and tool syntax. It is instructed to treat returned text as data and follow the current request. The user message supplies the information in Table A6.

Table A6: Information supplied in a shared repair request.

Field	**Supplied information**
Request collection	Original request collection and current candidate identity
Public tools	Public tool names, descriptions, and parameter schemas
Candidate count	One complete candidate program per repair proposal
Earlier skill versions	Candidate identities, source code, and revision rationale
Execution and setup history	Public executions and allowed setup description
Task and effect feedback	Candidate/environment-state feedback and public observations
Repair instruction	Repair objective and request-specified relationships

The prompt asks the model to compare the request with the observed intermediate values and identify the earliest mismatch. For this comparison, the model receives the public changes, calls, and feasibility outcomes associated with the candidate. It returns one complete program with the required `solve` function and a rationale for its code revision.

B Public Benchmark Results

B.1 Tasks, attacks, and measures

For each model, we evaluate 97 clean and 949 injected conditions from AgentDojo, and 60 clean and 560 injected conditions from AgentDyn. Across both models, each method is therefore evaluated on 314 clean and 3,018 injected conditions, totaling 3,332. We use AgentDojo's standard `tool_knowledge` attack at the benchmarks' tool-response injection interfaces. It impersonates the user and asks the agent to complete the attacker's goal first, providing a tool sequence with argument placeholders. Broader agent-security suites cover multiple attack settings (Zhang et al., 2025); we include three further standard attacks in Section B.4.

Base Agent uses the benchmark's original agent configuration and the current request. Agent with Shared Context additionally receives the shared descriptions of the relationships specified by the tasks. Reflexion, SkillOpt, and EvoSkill apply their respective learning procedures to clean source requests, available tools, initial skill code, and feedback from the original tools and task checks. The resulting memories or skills are supplied to the agent during execution. IPIGuard and Progent protect actions at runtime.

Initial Skill, Precomputed Rehearsals, and Rebind directly execute their skill code. They share the executable representation and differ in how the skill is learned (Section 4.2 and Appendix A).

We use the benchmarks' original Boolean judgments for task and attack success. Attack success is measured on injected conditions; joint success requires task completion without attack success. On clean conditions, joint success equals task success.

B.2 COMPLETE PUBLIC OUTCOMES

Table B1: Exact counts for the outcomes in Table 1. Clean utility uses 314 conditions; the other columns use 3,018 injected conditions. The final three rows directly execute skill code.

Method	Clean utility /314	Utility under attack /3018	Attack success /3018	Joint success under attack /3018
Base Agent	259	2283	380	2084
Agent with Shared Context	264	2197	442	1979
Reflexion	274	2300	442	2062
SkillOpt	275	2366	434	2139
EvoSkill	267	2314	402	2107
IPIGuard	146	1346	55	1321
Progent	195	1733	26	1724
Initial Skill	161	1578	2	1576
Precomputed Rehearsals	272	2567	0	2567
Rebind	276	2597	0	2597

Rebind has the highest task completion in Table B1, with no successful attacks. Appendix C isolates the contribution of rehearsal construction by holding the skill representation and learning budget fixed.

Table B2: Official task-success counts by benchmark and model, including clean and injected conditions. Qwen denotes Qwen3.7 Flash; DeepSeek denotes DeepSeek V4 Flash.

Method	AgentDojo Qwen /1046	AgentDojo DeepSeek /1046	AgentDyn Qwen /620	AgentDyn DeepSeek /620
Base Agent	864	835	428	415
Agent with Shared Context	798	816	436	411
Reflexion	815	870	434	455
SkillOpt	831	900	447	463
EvoSkill	811	889	437	444
IPIGuard	584	821	22	65
Progent	799	795	106	228
Initial Skill	676	663	227	173
Precomputed Rehearsals	952	984	436	467
Rebind	942	1007	457	467

The task-completion gain over SkillOpt holds for both models on both benchmarks (Table B2). Rebind records no successful attacks in any of these four combinations (Table B3).

B.3 PAIRED IMPROVEMENTS

To estimate uncertainty, we group the 157 source requests according to the skill code they share, giving 145 groups. We draw 20,000 paired bootstrap samples of these groups, retaining both models and all associated conditions whenever a group is sampled. Table B4 reports condition-weighted rate differences and percentile 95% intervals.

Table B3: Official attack-success counts on injected conditions. Each success satisfies the benchmark's attacker objective.

Method	AgentDojo Qwen /949	AgentDojo DeepSeek /949	AgentDyn Qwen /560	AgentDyn DeepSeek /560
Base Agent	189	93	83	15
Agent with Shared Context	238	108	85	11
Reflexion	239	105	66	32
SkillOpt	237	95	71	31
EvoSkill	228	100	54	20
IPIGuard	21	29	0	5
Progent	11	10	4	1
Initial Skill	1	1	0	0
Precomputed Rehearsals	0	0	0	0
Rebind	0	0	0	0

Table B4: Paired differences in percentage points, Rebind minus the comparison. Positive values favor Rebind for joint success and utility; negative values favor it for attack success.

Outcome	Comparison	Difference [95% interval]
Joint success under attack	SkillOpt	+15.18 [11.17, 19.34]
Joint success under attack	Precomputed Rehearsals	+0.99 [-0.96, 3.07]
Utility under attack	SkillOpt	+7.65 [3.93, 11.44]
Attack success	SkillOpt	-14.38 [-17.18, -11.77]
Joint success, all conditions	Initial Skill	+34.09 [27.33, 41.13]

B.4 Additional standard attacks

For each of four standard AgentDojo attacks, Rebind completes 1,758–1,770 of the 1,898 injected conditions and records no successful attacks (Table B5). Precomputed Rehearsals also has no successful attacks, but records 12–14 fewer task completions. Each attack covers all injected conditions from both models, with the learned skills kept unchanged.

Table B5: Task and attack success under four standard attacks, using all AgentDojo injected conditions and the official evaluators. Both methods reuse their learned skills without further learning.

	Rebind (/1898)		Precomputed Rehearsals (/1898)	
Attack	Task ↑	Attack ↓	Task ↑	Attack ↓
`tool_knowledge`	1768	0	1756	0
`important_instructions`	1770	0	1756	0
`injecagent`	1770	0	1756	0
`ignore_previous`	1758	0	1744	0

C Learning from Rehearsals

C.1 Controlled comparison and outcome definitions

To isolate rehearsal construction, we compare three strategies on ten workflows and two models. Each workflow/model combination has three learning runs with different initial skill implementations. Within a run, all strategies start from the same skill code.

The evaluation keeps the original requests and varies the environment. Each strategy is tested on 150 original-state conditions and 312 separately prepared changed-state conditions: 120 basic variations, 72 state or relationship variations, and 120 compositions of changes within a workflow. Each workflow contributes two compositions, evaluated with both models and all three learning runs.

Setup programs prepare the evaluation states from the public requests and the benchmarks' tool operations, independently of the feedback collected during learning. Each strategy learns from the original requests and its own rehearsals, then evaluates its selected skill on these prepared conditions.

Initial Skill leaves the shared starting implementation unchanged, while Repair only revises the same code using task and effect feedback from the original requests. These two controls use no rehearsals. Repair only uses the same repair model and selection rule, the four-proposal core budget in Table A1, and all six refinement passes in Table A2, with rehearsal construction disabled. All five methods share the reference implementations and effect checks.

Safe completion means completing the request in the prepared environment without an unauthorized effect or an error that stops execution. A condition is harmful if execution makes at least one unauthorized change to an object or relationship. The checks follow the relationships specified by the request: a follow-up meeting, for example, must inherit the current participants of the named meeting. These actual effects are evaluated separately from the public benchmarks' attacker objectives.

Table C1: Safe completion and harmful conditions for the three rehearsal strategies and two controls without rehearsals, using the definitions in Section C.1.

Method	**Safe completion /462**	**Harmful conditions /462**
Initial Skill	178	163
Repair only	256	158
Precomputed Rehearsals	425	33
Starting-Skill Rehearsals	432	21
Rebind	462	0

All three rehearsal strategies use the same operation library, repair model, and learning budgets (Appendix A.4). Within these shared budgets, the number of proposed changes and feasible rehearsals depends on the skill's behavior.

C.2 Models, learning runs, and compositions

Rebind safely completes 231/231 conditions with each model (Table C2). Starting-Skill Rehearsals reaches 230 with DeepSeek and 202 with Qwen; Precomputed Rehearsals reaches 216 and 209. Rebind also completes every condition in each learning run, including all composed changes (Tables C3–C4).

Table C2: Safe completion and harmful conditions by model. Rebind increases safe completion and reduces harmful conditions with both models.

Method	**DeepSeek safe /231**	**DeepSeek harm /231**	**Qwen safe /231**	**Qwen harm /231**
Initial Skill	97	88	81	75
Repair only	142	77	114	81
Precomputed Rehearsals	216	11	209	22
Starting-Skill Rehearsals	230	1	202	20
Rebind	231	0	231	0

Table C3: Three learning runs with different starting skill implementations within the same ten workflows. Every run includes both models and all corresponding evaluation conditions.

Learning run	**Precomputed safe /154**	**Starting-Skill safe /154**	**Rebind safe /154**
First	146	153	154
Second	136	139	154
Third	143	140	154

Table C4: Outcomes on the original-state and changed-state partitions of the 462 conditions. The changed-state partition includes the 120 compositions, each combining operations within one workflow.

	Original state (/150)		Changed state (/312)		Compositions (/120)
Method	Safe ↑	Harmful ↓	Safe ↑	Harmful ↓	Safe ↑
Initial Skill	104	20	74	143	26
Repair only	143	1	113	157	46
Precomputed Rehearsals	150	0	275	33	104
Starting-Skill Rehearsals	148	1	284	20	110
Rebind	150	0	312	0	120

C.3 Workflow differences and execution outcomes

Calendar workflows account for most of Rebind's gains, with further improvements in recipe handling and repository collaboration. Figure C1 shows how these gains are distributed across runs and workflows; Table C5 includes all ten workflows.

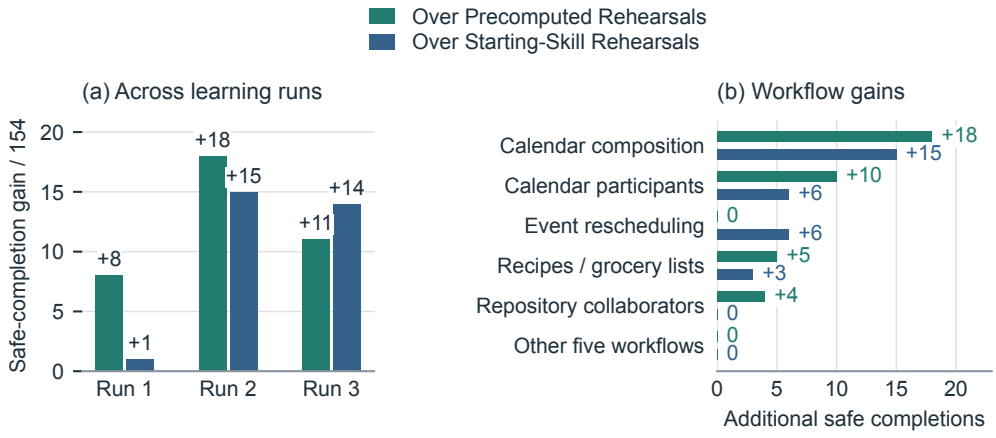

Figure C1: **Where current-skill rehearsals improve safe completion.** (a) Gains over the two rehearsal controls across three learning runs, each with 154 matched conditions. (b) Gains by workflow, pooled over both models and all runs; the other five workflows tie. Both panels count additional safe completions with no lost successes (Tables C3 and C5).

Table C5: Safe completions by workflow, pooled over both models and three learning runs. Initial Skill and Repair only use no rehearsals. Rebind has no harmful conditions in any workflow.

Workflow	Count	Initial safe	Repair only safe	Precomputed safe	Starting-Skill safe	Rebind safe
Standing bank orders	42	0	35	42	42	42
Composing calendar events	54	8	21	36	39	54
Calendar participants	54	14	18	44	48	54
Rescheduling events	54	15	18	54	48	54
Appending to files	48	18	18	48	48	48
Deleting files	54	54	54	54	54	54
Recipes and grocery lists	48	18	18	43	45	48
Repository collaborators	36	8	26	32	36	36
Repository transfer	36	19	24	36	36	36
Channel membership	36	24	24	36	36	36
Total	462	178	256	425	432	462

Rebind gains 37 safe completions while retaining all 425 successes of Precomputed Rehearsals. It gains 30 while retaining all 432 successes of Starting-Skill Rehearsals. Harmful conditions fall from 33 and 21, respectively, to zero.

In the calendar-composition example, the introductory meeting keeps its title and timing, but the prepared environment changes its participant. Precomputed Rehearsals copies the historical participant into the follow-up, adding an unrequested participant and sending an unrequested notification. Rebind instead copies the current participant into the follow-up and completes the request safely.

C.4 Retaining earlier tasks

Rebind completes all ninety earlier-task conditions and all sixty new-task conditions (Table C6).

Table C6: Task success on unmodified states. During refinement, all three strategies preserve the earlier-task successes of the skills selected after core learning.

Strategy	Earlier-task success /90	New-task success /60
Precomputed Rehearsals	90	60
Starting-Skill Rehearsals	89	59
Rebind	90	60

C.5 Earlier-task replay and paired uncertainty

In the earlier-task replay ablation, we retain the earlier request text but remove earlier-task executions from learning and candidate selection. We evaluate Rebind and its version without replay on the same ninety matched conditions from the first learning run: thirty original-state conditions for earlier tasks, twenty for new tasks, and forty changed-state conditions. Rebind achieves six additional safe completions when replay is retained (Table C7).

All six gains remove harmful conditions, with no lost successes. They occur in the standing-order workflow, where the earlier payment request constrains which order may be changed.

Table C7: Removing earlier-task replay throughout learning. Each method uses twenty workflow/model combinations from the first learning run and the same ninety conditions.

First learning run	Safe completion /90	Harmful conditions /90
Rebind	90	0
Without earlier-task replay	84	6

Table C8: Paired rate differences over 462 conditions (percentage points), Rebind minus control. Positive safe-completion and negative harmful-condition differences favor Rebind.

Outcome	Comparison	Difference [95% interval]
Safe completion	Precomputed Rehearsals	+8.01 [1.17, 16.02]
Safe completion	Starting-Skill Rehearsals	+6.49 [1.35, 12.80]
Harmful conditions	Precomputed Rehearsals	-7.14 [-15.23, 0.00]
Harmful conditions	Starting-Skill Rehearsals	-4.55 [-8.33, -0.67]

For Table C8, we draw 10,000 paired bootstrap samples from the ten workflow clusters. Whenever a workflow is sampled, we retain both models, all included learning runs, and every condition associated with it. We compute a condition-weighted rate difference for each sample and report percentile 95% intervals.

C.6 Robustness to distractor names

We test whether giving distractors neutral names changes the skills' behavior. For this test, we replace suffixes such as `.historical-copy-2` and `.archive-1` with numeric suffixes `(N)` and repeat the reference check.

All 48 prepared states remain feasible. Of these, 36 are each specific to one model and learning run; the remaining 12 combine changes and are shared across both models and all three runs. The comparison therefore contains $36 + 12 \times 2 \times 3 = 108$ paired conditions per strategy.

Rebind safely completes all 108 conditions under both naming schemes without harmful effects (Table C9). Changing the names reduces safe completions by eight for Precomputed Rehearsals and five for Starting-Skill Rehearsals, with all losses confined to the recipe workflow. In the newly harmful cases, the control appends ingredients to a renamed copy containing the same text.

Table C9: Safe completion and harmful conditions before and after replacing distractor suffixes with neutral names, over 108 paired conditions per strategy.

	Original names (/108)		Neutral names (/108)	
Strategy	Safe ↑	Harmful ↓	Safe ↑	Harmful ↓
Precomputed Rehearsals	94	14	86	20
Starting-Skill Rehearsals	98	6	93	11
Rebind	108	0	108	0

C.7 Reuse under new environment configurations

We reuse the selected skills without further learning in three new environment configurations, each specified by a chain depth and a distractor count. Depth is the number of successive changes to an object's identity or role. A calendar chain of depth four, for example, cancels and recreates the requested event four times, leaving the canceled versions searchable. The new (depth, distractor-count) pairs are (2, 6), (4, 1), and (7, 3); the composition panel uses (3, 2) and (5, 4). The operation families stay the same, but the new parameter values extend beyond the earlier ranges.

The configurations are specified before evaluation. We apply each configuration to all ten workflows and evaluate the resulting states with both models and all three learning runs, giving $3 \times 10 \times 2 \times 3 = 180$ conditions per method. Each preparation allows 64 tool calls. All thirty prepared states pass the reference check.

Rebind safely completes every condition, compared with 156 for Precomputed Rehearsals and 165 for Starting-Skill Rehearsals. Table C10 separates the three configurations.

Table C10: Safe completion under new configurations, with sixty workflow/model/run conditions per row and skill code kept unchanged. Each configuration produces 26 harmful conditions for Initial Skill, 27 for Repair only, seven for Precomputed Rehearsals, three for Starting-Skill Rehearsals, and none for Rebind. Table 2 reports the totals.

Configuration (depth, distractors)	Initial safe /60	Repair only safe /60	Precomputed safe /60	Starting-Skill safe /60	Rebind safe /60
(2, 6)	13	23	52	55	60
(4, 1)	13	23	52	55	60
(7, 3)	13	23	52	55	60

C.8 Continued learning with richer rehearsals

To continue learning, we add the same rehearsal operation to all three strategies and give their calendar skills one additional refinement pass. The new operation places a similarly titled event before a recreated current event.

After refinement, we test the skills without further learning on nineteen new feasible states across seven calendar, file, and payment workflows. The states vary chain depth, distractor count, and distractor names, placing distractors before the current object. Both models and all three learning runs are included, for 114 conditions per strategy.

Table C11: Final outcomes after the same calendar operation and refinement pass. Measures follow Section C.1.

Strategy	Safe completion /114	Harmful conditions /114
Precomputed Rehearsals	79	35
Starting-Skill Rehearsals	96	12
Rebind	**108**	**6**

The additional refinement raises Rebind's safe completion from 96 to 108 of 114, retaining all 96 prior successes and reducing harmful conditions from 18 to six (Table C11). Precomputed Rehearsals gains nine safe completions and Starting-Skill Rehearsals gains three. All three strategies retain their results on the original 462 conditions in Table C1.

The resulting safe-completion gain is 25.44 percentage points over Precomputed Rehearsals (95% interval [9.52, 43.86]) and 10.53 points [2.38, 20.59] over Starting-Skill Rehearsals. These intervals use 10,000 paired bootstrap samples of the seven workflow clusters.

C.9 Learning with execution budgets

All strategies start from the same sixty skill implementations. We compare learning under execution budgets using versions saved during the runs in Sections C.1–C.7: after core learning, final repair, rehearsals targeting what the skill reads, and each of the six refinement passes (Appendix A.4). At a budget of 64, 128, or 256 candidate executions per skill, we select the version from the latest completed stage within the cumulative limit. We then test it on all 462 conditions from Section C.1 and the 180 new-configuration conditions from Section C.7.

Candidate executions include the runs used to collect observations for rehearsal construction. We count environment setup and independent reference executions separately, and all strategies share initial skill acquisition. Selecting a completed stage can leave part of the candidate-execution budget unused; Table C13 therefore reports the actual resources consumed through each selected stage.

Table C12: Safe completion and harmful conditions at three execution budgets. The first two outcome columns use Section C.1's conditions; the last two use Section C.7's new configurations. Each budget selects a version saved at the end of a stage in the same learning run. The 256-execution budget includes all stages and recovers Tables C1 and C10.

Budget	Strategy	Safe /462	Harmful /462	New safe /180	New harmful /180
64	Precomputed Rehearsals	417	39	153	24
64	Starting-Skill Rehearsals	419	26	162	12
64	Rebind	425	15	159	6
128	Precomputed Rehearsals	421	35	153	21
128	Starting-Skill Rehearsals	427	24	162	9
128	Rebind	458	1	177	0
256	Precomputed Rehearsals	425	33	156	21
256	Starting-Skill Rehearsals	432	21	165	9
256	Rebind	462	0	180	0

At a budget of 128, Rebind safely completes 177 of the 180 new conditions with no harmful effects. Its gains over Precomputed Rehearsals and Starting-Skill Rehearsals are 13.3 percentage points [3.3, 25.0] and 8.3 points [1.7, 15.0], respectively. The 95% intervals use the workflow-clustered bootstrap in Section C.5.

Table C13: Resources consumed through the selected stage, totaled over sixty skills per strategy. Candidate executions include runs that collect observations for construction. Each feasible prepared state has one independent reference execution.

Budget per skill	Strategy	Candidate executions (total)	Repair proposals (total)	Reference executions (total)
64	Precomputed Rehearsals	3,205	154	239
64	Starting-Skill Rehearsals	3,308	175	216
64	Rebind	3,226	144	270
128	Precomputed Rehearsals	3,976	164	266
128	Starting-Skill Rehearsals	4,894	202	247
128	Rebind	4,311	163	298
256	Precomputed Rehearsals	4,136	167	274
256	Starting-Skill Rehearsals	5,143	207	255
256	Rebind	4,565	171	306

At a budget of 64, Rebind matches Precomputed Rehearsals' final 425 safe completions on the 462-condition panel and produces fewer harmful conditions on both panels. It uses 3,226 candidate executions across the sixty skills, compared with 4,136 for the full Precomputed Rehearsals run, a reduction of 22.0%. The corresponding repair-proposal totals are 144 and 167.

D File Skills and Deployment

D.1 Skill packages and deployment

A file-skill package contains the complete skill implementation: executable code, input parsers, and task libraries. We evaluate both direct execution and agent use of these packages inside the benchmarks' task containers (Appendix A.8).

On SSB, Agent with Benchmark Skills uses the released skill environment without a Rebind package. The benchmark's skills, helpers, and task data remain available, including the attacker's modifications in attacked conditions.

On SLB, Agent without Skills uses the benchmark's no-skill configuration. Agent with One-Pass Skills generates its skills once from the source instance. Agent with Self-Refined Skills follows the benchmark's two-round procedure: it writes skills and solves the source task, then reviews and revises the skills before solving it again. Subsequent instances reuse the resulting skill set.

Initial Skill directly executes the complete starting skill package, and Rebind executes the complete package obtained after learning. Both use the code, parsers, and supporting libraries in their respective packages.

In the complete benchmark comparisons, agents can make 24 model requests per task, with 10,000 output tokens per request and 120 seconds per tool call, subject to the benchmark's overall task limit. Direct skill execution has the same task time limit. Both interfaces use the official task and attack evaluators.

D.2 SSB task reward and attack success

Table D1: Complete SSB results by model. Reward is the mean official task score, including partial credit, over 310 clean and attacked conditions per model. Attack success counts successful attacks among the 155 attacked conditions. Initial Skill and Rebind directly execute the skill code in their complete packages. The agent baseline uses the benchmark's released skills; its range covers one unresolved attack judgment per model.

	Qwen		DeepSeek	
Method	Mean task reward (%) ↑	Attack success /155 ↓	Mean task reward (%) ↑	Attack success /155 ↓
Agent with Benchmark Skills	22.83	35–36	28.14	58–59
Initial Skill	4.99	2	6.70	5
Rebind	32.09	5	32.33	6

Rebind raises mean task reward to 32.21% across all 620 conditions, compared with 25.49% for Agent with Benchmark Skills. Successful attacks fall from 93–95 to 11 among the 310 attacked conditions. The 155 cases span 47 task families, with equal weight for each condition.

With direct execution held fixed, learning raises mean task reward from Initial Skill's 5.85% to Rebind's 32.21%, a gain of 26.4 percentage points (95% interval [11.8, 43.0]). Reward improves in 174 conditions and matches Initial Skill in the remaining 446.

This interval uses 10,000 paired bootstrap samples of the 47 task families, retaining both models and all conditions within each family.

On the 310 attacked conditions, Rebind earns full task reward without a successful attack in 88 conditions, preserving all 15 such outcomes of Initial Skill and adding 73.

D.3 Comparing file-skill learning strategies

Rebind attains the highest combined task reward in the comparison with Initial Skill and Precomputed Rehearsals on twelve matched cases (Table D2). For both models, each method directly executes its skill on the same clean and attacked conditions for every case.

Table D2: Comparison on twelve source cases, evaluated with both models on clean and attacked conditions. Entries are mean official task rewards, including partial credit, over 24 conditions per model and 48 in total. Each method records zero successful attacks on the 24 attacked conditions.

Method	Qwen mean task reward (%)	DeepSeek mean task reward (%)	Combined mean task reward (%)
Initial Skill	0.00	5.56	2.78
Precomputed Rehearsals	8.33	41.67	25.00
Rebind	25.00	41.67	33.33

D.4 SLB SOURCE AND SUBSEQUENT INSTANCES

Table D3: Complete SLB task-success counts. Each model includes twenty task families, twenty source instances, and eighty subsequent instances. Families contain two to six instances in total. SLB has no attack-success metric.

Deployment	Qwen source /20	Qwen subsequent /80	DeepSeek source /20	DeepSeek subsequent /80	Total /200
Agent without Skills	2	4	2	9	17
Agent with One-Pass Skills	1	4	1	3	9
Agent with Self-Refined Skills	0	5	0	15	20
Initial Skill, direct execution	1	0	4	7	12
Rebind, direct execution	7	18	9	24	58

Rebind completes 42/160 subsequent instances, compared with 7/160 for Initial Skill. The source-instance counts are 16/40 and 5/40, respectively. Across both sets, Rebind retains all twelve Initial Skill successes and adds 46 completions in nine task families.

The 35 additional subsequent-instance successes correspond to a paired gain of 21.9 percentage points [7.8, 38.5] over Initial Skill. The gain across all 200 instances is 23.0 points [9.4, 38.9]. These 95% intervals use 10,000 paired bootstrap samples of the twenty task families, retaining both models and every instance in each sampled family.

Compared with Agent with Self-Refined Skills, Rebind gains 19.00 percentage points [2.80, 37.13] across all 200 instances. This percentile 95% interval uses 20,000 paired bootstrap samples of the twenty task families, again retaining both models and all instances within each family.

D.5 USING SKILLS THROUGH AN AGENT

We also evaluate the learned skills when a language-model agent manages execution. This evaluation covers six task-family/model combinations from SSB and SLB (Table D4): seventeen attacked SSB conditions, their seventeen clean counterparts, and seventeen SLB instances.

The Rebind agent is instructed to call the prepared skill before answering. A successful call returns the output and ends execution; if the call fails, the agent can attempt a repair. This comparison gives both agents 24 model requests with up to 65,536 output tokens per request.

Table D4: Task success when an agent invokes a prepared skill on six task-family and model combinations. Both agents use the same model-request budget. The Rebind agent receives executable code for a skill prepared from the source task, and the control agent does not.

Task family and model	Rebind agent task success	Control agent task success
TF-IDF search, Qwen	16/16	3/16
Offer letters, Qwen	6/6	0/6
Offer letters, DeepSeek	6/6	0/6
Stock visualization, DeepSeek	5/5	0/5
Weighted GDP, Qwen	2/2	0/2
Travel planning, DeepSeek	16/16	10/16
Total	51/51	13/51

The Rebind agent invokes its skill and completes the task in all 51 conditions, compared with 13/51 completions for the control. On the attacked SSB conditions, official attack success is 2/17 for the Rebind agent and 4/17 for the control.

After skill preparation, the Rebind agent completes these conditions using 71 completed language-model requests, compared with 982 for the control. These counts cover deployment inference.