RRU: IMPROVING AGENT SKILL GENERATION WITH REFLECT–REPLAY–UPDATE

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills extend language agents with workflows and knowledge beyond their pre-trained capabilities. Existing skill generation frameworks adopt a **Reflect-then-Update** approach: the agent solves a batch of tasks, reflects on the rollouts, and updates the skill to capture recurring patterns and resolve conflicts. This approach can let rollouts from a single batch remove an instruction supported by earlier batches (*overwriting*) or become an instruction applied to all tasks (*over-generalization*). These failures arise because each update conditions only on the current batch, as earlier rollouts are often discarded after reflection. We introduce **Reflect–Replay–Update (RRU)**, which improves grounding by maintaining compact records of the rollouts that provide evidence supporting each instruction and replaying these rollouts during skill updates. With replay, an update can revise an earlier instruction rather than remove it, and generalize a new instruction only when earlier batches support it. On AppWorld, SpreadsheetBench, and Legal Agent Bench with two open-weight models, RRU outperforms strong baselines (Trace2Skill, SkillOpt, and Combee) by an average of 8.5% on AppWorld and 5.6% on SpreadsheetBench across the two models, and by 3.7% and 1.2% on Legal Agent Bench with Qwen and Gemma. Ablations show that replay improves accuracy by 5.0% on average and reduces variation across seeds in three of four settings, and that compact records reach higher accuracy than complete rollouts in three of four settings and use fewer tokens in three of four settings (up to 30%).

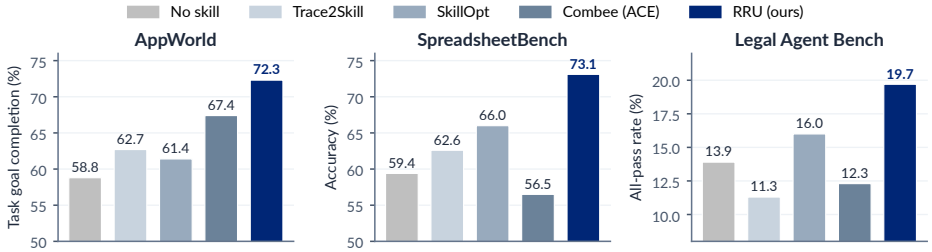


Figure 1: **Overall performance.** RRU outperforms strong skill generation baselines across tool use, spreadsheet, and legal tasks (Qwen3.6-35B-A3B).

1 INTRODUCTION

Agent skills (Anthropic, 2025) extend language agents with workflows and knowledge beyond their pre-trained capabilities. A skill consists of a `SKILL.md` file containing instructions for the agent, together with optional scripts and reference files loaded when needed (Anthropic, 2025). Unlike fine-tuning, skills leave model weights unchanged, and remain readable and editable by humans. In practice, developers write skills iteratively: they run the agent on target tasks, inspect failed rollouts, and add instructions that correct recurring mistakes, while merging overlapping instructions and resolving conflicts (Anthropic, 2025). Because this manual process does not scale across many domains, recent frameworks generate skills from the agent's own rollouts (Zhang et al., 2026a; Ni et al., 2026; Yang et al., 2026; Li et al., 2026a).

These frameworks often follow a **Reflect-then-Update** approach: the agent solves a batch of training tasks with the current skill, reflects on the rollouts, and updates the skill to capture recurring

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.

patterns and resolve conflicts. We observe that, in this approach, rollouts from a single batch can remove an instruction supported by earlier batches (*overwriting*; Figure 3) or become an instruction applied to all tasks (*over-generalization*; §2). In an extreme case, these failures reduce accuracy below that of the model without a skill (Figure 2; §5.2 reports the variation across seeds). These failures arise because each update conditions only on the current batch, as the rollouts of earlier batches are often discarded after reflection: when revising an instruction, the update has only the new reflections and the current skill, which summarizes earlier reflections without their rollouts (Ni et al., 2026; Yang et al., 2026; Li et al., 2026a). Intuitively, human developers do not revise an instruction from the latest batch alone: they first revisit earlier rollouts to ground the update decision.

Following this intuition, we introduce **Reflect–Replay–Update (RRU)** (§4), which improves the Reflect-then-Update approach by maintaining compact records of the rollouts that provide evidence supporting each instruction and replaying these rollouts during skill updates. The records of an instruction form its *support set* (§4.2). Each step consists of three stages (Figure 4). (i) *Reflect*: as in prior frameworks, the agent solves the batch and a model reflects on the rollouts; each rollout is stored as a compact record that keeps its task, actions, outcome, and reflection but omits the environment observations (§4.2, §4.2.1). (ii) *Replay*: RRU retrieves the records from the new batch together with the support sets of the instructions that the update may revise; because the records are compact, the support sets of many instructions fit within a single update context (§4.2.2). (iii) *Update*: RRU adds instructions for new recurring patterns and restructures existing ones with these records in context, so that an earlier instruction is revised rather than removed, and a new instruction is generalized only when earlier batches support it; each instruction in the updated skill cites the records that support it, which later updates replay (§4.2.3).

We evaluate RRU on three benchmarks covering tool use (AppWorld; Trivedi et al., 2024), spreadsheet manipulation (SpreadsheetBench; Ma et al., 2024), and legal drafting (Legal Agent Bench; Harvey AI, 2026), using Qwen3.6-35B-A3B and Gemma-4-26B-A4B-it (Gemma Team, 2026) as agent models (§5). RRU outperforms strong baselines, including Trace2Skill (Ni et al., 2026), SkillOpt (Yang et al., 2026), and Combee (Li et al., 2026a), by an average of 8.5% on AppWorld and 5.6% on SpreadsheetBench across the two models, and by 3.7% and 1.2% on Legal Agent Bench with Qwen and Gemma (Figure 1, Table 1). Ablations show that replay improves accuracy by 5.0% on average and reduces variation across seeds in three of four settings (§5.2, Table 2), and that compact records reach higher accuracy than complete rollouts in three of four settings and use fewer tokens in three of four settings (up to 30%; Table 3). Finally, we study the potential of skill generation with and without fine-tuning access, and find that without fine-tuning access, RRU outperforms two epochs of GRPO with 7.8–8.0$\times$ fewer rollouts, while with fine-tuning access, adding its skill to the GRPO checkpoints improves them by 11.1–14.0% after each epoch (§5.3, Figure 5). Our contributions are:

- We identify two failure modes of the Reflect-then-Update approach, *overwriting* and *over-generalization*, in which rollouts from a single batch, seen without the earlier rollouts, remove or over-extend instructions (§2; Figures 2 and 3).
- We propose RRU (§4; Figure 4), which maintains compact records of the rollouts that provide evidence supporting each instruction and replays these rollouts during skill updates, so an update revises instructions rather than removing them and generalizes a new one only when earlier batches support it.
- We show that RRU outperforms Trace2Skill, SkillOpt, and Combee on three benchmarks with two models, that replay improves accuracy and stability across seeds, and that its skills can be added to models fine-tuned with reinforcement learning (§5; Tables 1–3, Figure 5).

2 Background and Motivation

2.1 Reflect-then-Update

Existing skill generation frameworks (Zhang et al., 2026a; Ni et al., 2026; Yang et al., 2026; Li et al., 2026a) share a Reflect-then-Update approach (formalized in §4.1). The agent solves a batch of training tasks with the current skill, a model reflects on the individual rollouts and the batch, and an update revises the skill from these reflections. Like a developer maintaining a skill by hand, the update identifies recurring patterns across rollouts, merges overlapping instructions, and resolves

conflicts. The frameworks differ in how they aggregate reflections and whether they validate an edit, but they discard the rollouts after reflection. Each update therefore conditions only on the new reflections and the current skill.

2.2 Two Failures of the Reflect-then-Update Approach

In our experiments (§5), we observe two failure modes in which rollouts from a single batch remove or over-extend instructions supported by earlier batches.

Overwriting. When a new rollout contradicts an instruction learned from earlier batches, the update replaces the instruction instead of restricting its scope. Without the rollouts that support the instruction, the update is unable to distinguish an exception from a counterexample. For example, on spreadsheet manipulation tasks (SpreadsheetBench; Ma et al., 2024), a skill learns to write a formula when a task asks for one and a computed value otherwise. After one failed task that asks to modify a formula, the update rewrites the instruction to prefer values for all tasks, although earlier rollouts show that tasks asking for a formula need one (Figure 3). A distinction learned in earlier batches is lost in a single update.

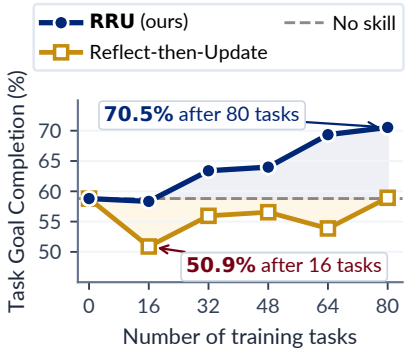


Figure 2: **Effect of the two failure modes on AppWorld** (Qwen3.6-35B-A3B, one seed). Without replay, the first update lowers task goal completion from 58.8% to 50.9%, below the model without a skill; with replay, it reaches 70.5%.

Over-generalization. Conversely, the update may turn a pattern in the failures of a single batch into an instruction applied to all tasks, although rollouts from earlier batches contradict it; the instruction then breaks tasks that were previously solved. Conditioning only on the current batch, the update is unable to verify whether the pattern extends beyond that batch. For example, on tasks that operate everyday apps through their APIs (AppWorld; Trivedi et al., 2024), the update turns one pair of rollouts on liking a payment transaction into an instruction to call the corresponding API for every action verb in a task. Following it, on four test tasks the agent repeatedly adds funds to complete a transfer blocked by insufficient balance instead of stopping. Baseline skills also fall below the model without a skill in some settings (§5.1).

In an extreme case, these failures lower accuracy below that of the model without a skill (Figure 2); SpreadsheetBench shows a similar pattern (Appendix E).

3 Related Work

Agent Skills, Memory, and Continual Learning. Acquiring and persisting task knowledge without parameter updates is a central problem for language-model agents (Weng, 2026). One direction focuses on agent memory, storing episodic or semantic facts across interactions (Packer et al., 2023; Xu et al., 2026). MemGPT (Packer et al., 2023) manages bounded context windows via tiered memory hierarchies, while A-Mem (Xu et al., 2026) organizes past interactions into a network of linked notes; other work reuses past trajectories, workflows, or plans at inference time (Zhao et al., 2024; Wang et al., 2024; Fang et al., 2026; Ouyang et al., 2026; Zhang et al., 2026b). Complementary to factual memory, *agent skills* package procedural workflows, tool-use conventions, and error-recovery strategies into structured markdown documents or skill directories (Anthropic, 2025; Li et al., 2026b; Jiang et al., 2026). Prior work has framed related artifacts as cumulative playbooks (Zhang et al., 2026a), dynamic cheatsheets (Suzgun et al., 2026), or executable tool libraries (Wang et al., 2023). Unlike conversational memory, which tracks user- or session-specific facts, agent skills provide reusable procedural instructions that guide the agent across tasks. We study how such procedural guidance can be updated and maintained as agents encounter new tasks.

Automated Skill Generation and Maintenance. Because manual skill authoring does not scale to diverse tasks, recent work has focused on generating skills automatically from agent experience. Most methods follow an iterative reflection loop: agents attempt tasks, analyze failures, and revise

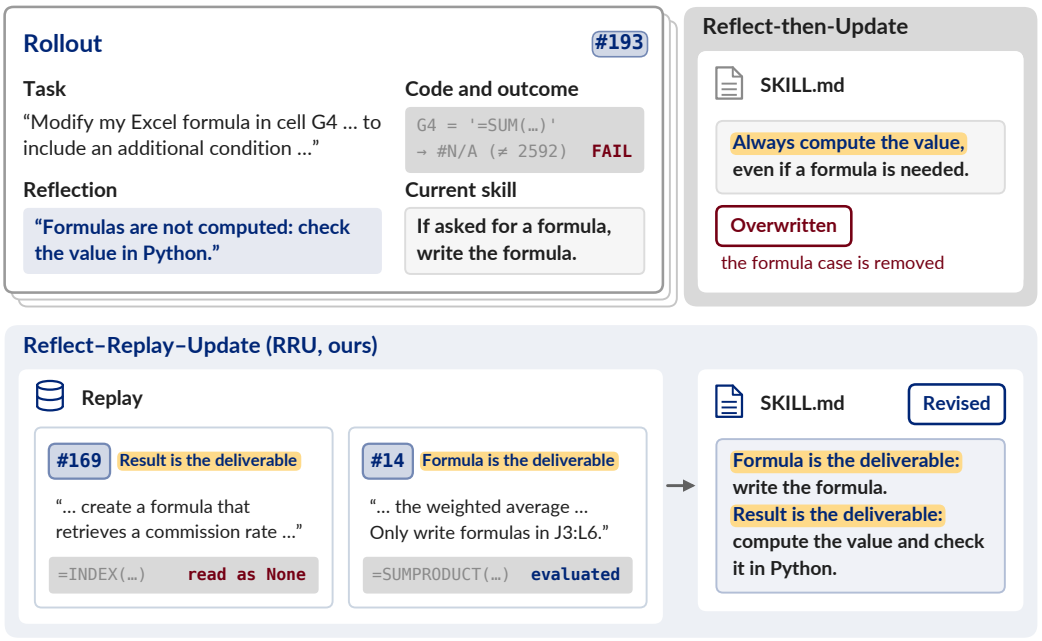


Figure 3: **An example of how replay alleviates overwriting.** Given a failed SpreadsheetBench rollout that asks to modify a formula and its reflection (top left), the update without replay (top right) writes computed values for all tasks, removing the case in which the formula is the deliverable. With replay (bottom), the update retrieves two earlier records that support the current instruction, which show that a formula is evaluated when it is the deliverable and read as None when the result is the deliverable, and revises the instruction into these two cases.

their memory (Shinn et al., 2023) or a shared document (Zhang et al., 2026a; Li et al., 2026a). ACE (Zhang et al., 2026a) grows a playbook by appending reflections, and Combee (Li et al., 2026a) extends it with a map-reduce stage that grows the playbook from parallel task attempts. Other methods iteratively refine modular skill files: Trace2Skill (Ni et al., 2026) extracts procedural lessons from trajectories into skill entries; SkillOpt (Yang et al., 2026) refines instructions using execution feedback and validation rollouts, and re-runs a sample of tasks at the end of each epoch; EvoSkill (Alzubi et al., 2026) searches over agent programs; and SkillX (Wang et al., 2026) organizes instructions into multi-tiered hierarchies. Further methods optimize prompts and other textual variables (Khattab et al., 2023; Opsahl-Ong et al., 2024; Yuksekgonul et al., 2024; Agrawal et al., 2026). However, most existing systems discard the rollouts after reflection and maintain skills as text alone. When new rollouts contradict an existing instruction, the update does not condition on the rollouts from which the instruction was derived, which leads to overwriting and over-generalization (§2.2). In contrast, RRU maintains compact records of the rollouts that provide evidence supporting each instruction and replays these rollouts during skill updates (§4.2).

Connection to Expert Systems and Truth Maintenance. Agent skills can be viewed as a natural-language form of expert systems: the skill document serves as an evolving knowledge base that specializes a general model without gradient updates, and new instructions can conflict with existing ones. Classical truth maintenance systems resolved contradictions by attaching explicit justifications to assertions and tracing underlying dependencies (Doyle, 1979; De Kleer, 1986). RRU applies this principle to agent skills: it links instructions to the records that support them and replays these records during updates rather than revising from text alone.

4 METHOD

RRU addresses overwriting and over-generalization (§2) by maintaining compact records of the rollouts that provide evidence supporting each instruction and replaying these rollouts during skill

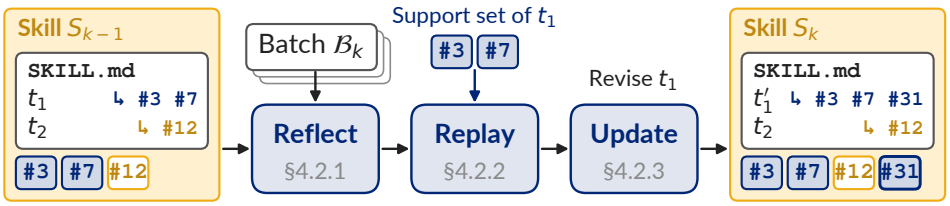


Figure 4: **Overview of the RRU framework.** Each instruction in the skill cites the records that support it. At step k, M solves the batch $\mathcal{B}_k$ with S_{k-1}, and **Reflect** stores a compact record of each rollout, including its reflection; **Replay** retrieves these records and the support sets of the instructions that the update may revise; **Update** edits S_{k-1} into S_k with the replayed records in context. In the example, Replay retrieves the support set of t_1 (#3, #7), and Update revises t_1 into t_1', adding the new record #31 to its support set.

updates. In §4.1, we formalize the skill generation problem and the Reflect-then-Update approach. In §4.2, we introduce RRU, its three stages (Figure 4), and how records are compactly represented.

4.1 Problem Formulation

Setting. Let M be a frozen language model that acts as an agent in an interactive environment and solves tasks x. The agent is conditioned on a skill S, a folder whose `SKILL.md` file holds a set of natural-language instructions $t \in S$ that the agent reads before every task, and optional files that it reads on demand (Anthropic, 2025). A *rollout* τ_x is the complete interaction on a task x; the environment scores it with $r(\tau_x) \in [0, 1]$ and returns textual feedback $f(\tau_x)$. Skill generation constructs S from the agent's rollouts on a training set $\mathcal{D}_{\text{train}}$ of tasks, without updating the parameters of M. We evaluate S on a disjoint test set $\mathcal{D}_{\text{test}}$, whose tasks are never seen during training, by the average score. Reflection and skill updates also prompt a frozen model M_U, which may differ from M (§5).

Reflect-then-Update. Existing frameworks (§2.1; generate–reflect–update in Li et al., 2026a) learn S from an empty skill S_0 over batches $\mathcal{B}_1, \mathcal{B}_2, \ldots$ of tasks from $\mathcal{D}_{\text{train}}$. At step k, M generates a rollout τ_x for each task $x \in \mathcal{B}_k$ with the current skill S_{k-1}. M_U generates reflections $\mathcal{R}_k$ from $\{(\tau_x, r(\tau_x), f(\tau_x))\}_{x \in \mathcal{B}_k}$ and updates the skill with an update U:

$$S_k = U\big(S_{k-1},\ \mathcal{R}_k\big). \tag{1}$$

U thus conditions only on $\mathcal{R}_k$ and S_{k-1}: when U revises an instruction $t \in S_{k-1}$, it does not condition on the earlier rollouts from which t was derived (Figure 3).

4.2 RRU: Reflect–Replay–Update

RRU improves the Reflect-then-Update loop by replaying the records that support each instruction during skill updates. A *record* $\rho(\tau_x)$ is a compact representation of the rollout τ_x (defined below). RRU keeps with each instruction t its *support set* $E(t)$, the records of the rollouts that support t. Let $E(S_{k-1}) = \bigcup_{t \in S_{k-1}} E(t)$ denote the union of the support sets of the current instructions. RRU updates the skill as

$$S_k = U\big(S_{k-1},\ \mathcal{R}_k,\ \underline{E(S_{k-1})}\big). \tag{2}$$

Compared with Eq. (1), U also conditions on the records that support each instruction (underlined), which addresses the failures in §2.2; in practice, each operation o of the update retrieves only the records relevant to it: those of the tasks that exhibit its pattern and $E(I(o))$ (§4.2.2). Each step consists of three stages (Figure 4; prompts in Appendix D): *Reflect* generates $\mathcal{R}_k$; *Replay* retrieves, for each operation that M_U proposes from $\mathcal{R}_k$, the records relevant to it; and *Update* applies these operations, maintaining the support set of every instruction it changes.

Representation of records. An important design choice is what a record keeps. A rollout interleaves actions with environment observations, which can be long for long-running agents. We implement the record $\rho(\tau_x)$ as the task, the actions, the outcome ($r(\tau_x)$ and $f(\tau_x)$), and the reflection (§4.2.1), omitting the observations. Empirically, a record is 12.6% and 11.4% of the rollout size on AppWorld and SpreadsheetBench, and replaying records reaches higher accuracy than replaying

complete rollouts in three of four settings, with fewer tokens in three of four settings (§5.2). RRU stores each record as a file in the skill folder, and each instruction cites the records in its support set.

4.2.1 REFLECT

RRU follows the batch-level reflection of prior frameworks (Ni et al., 2026; Li et al., 2026a). M solves each task $x \in \mathcal{B}_k$ with S_{k-1}. For each rollout τ_x, M_U writes a reflection on $r(\tau_x)$, $f(\tau_x)$, and how S_{k-1} contributed to the outcome, and RRU stores the task, actions, outcome, and reflection as the record $\rho(\tau_x)$. M_U then reads these reflections across the batch and generates $\mathcal{R}_k$: the *recurring patterns*, i.e., failures or procedures seen in several rollouts, each with the tasks that exhibit it.

4.2.2 REPLAY

Given $\mathcal{R}_k$, M_U proposes operations on S_{k-1}: each operation o either adds an instruction or edits existing instructions to deduplicate them, resolve a conflict, or capture a new recurring pattern. Let $I(o) \subseteq S_{k-1}$ denote the existing instructions that o edits or overlaps, as identified by M_U. Before M_U applies o, Replay renders in context the records of the tasks that exhibit its pattern and the support sets $E(I(o)) = \bigcup_{t \in I(o)} E(t)$. Replay reads stored records and re-executes no rollout.

4.2.3 UPDATE

M_U applies each operation o with these records in context, and RRU updates the support sets: an added instruction takes the records of the tasks that exhibit its pattern, an edited instruction keeps the records in $E(I(o))$ and the records of $\mathcal{B}_k$ that support it, and an unchanged instruction keeps its support set. Replay informs two decisions of the update (§2.2). First, when new records contradict an instruction $t \in I(o)$, $E(t)$ shows where t holds, so M_U can restrict t to these cases rather than remove it, which addresses overwriting (Figure 3). Second, when a new pattern overlaps instructions in $I(o)$, their support sets can show where the pattern does not hold, so M_U can limit the new instruction to the cases its records support, which addresses over-generalization.

5 EXPERIMENTS

We compare RRU with skill generation baselines (§5.1), ablate replay and the representation of a record (§5.2), and compare skill generation with reinforcement learning (§5.3).

Benchmarks. We use **AppWorld**, used by ACE and Combee (Zhang et al., 2026a; Li et al., 2026a), **SpreadsheetBench**, used by Trace2Skill and SkillOpt (Ni et al., 2026; Yang et al., 2026), and **Legal Agent Bench** for long-form expert writing: (1) **AppWorld** (Trivedi et al., 2024): interactive tool orchestration across nine daily applications (e.g., Spotify, Venmo, Gmail). We train skills on 90 tasks and evaluate on the 168-task *test-normal* split, reporting standard metrics: Task Goal Completion (**TGC**) and Scenario Goal Completion (**SGC**, where a scenario passes only if all its constituent tasks succeed). (2) **SpreadsheetBench** (Ma et al., 2024): code execution and complex workbook manipulation using `openpyxl` (200 training and 144 test tasks). (3) **Legal Agent Bench** (Harvey AI, 2026): legal memo drafting and agreement negotiation; we use its two memo task kinds (playbook escalation and term negotiation), keep the 96 tasks with the lowest cost in a run without a skill, and split them into 48 training and 48 test tasks so that no combination of contract type and task kind appears in both splits. Each Legal Agent Bench task is evaluated by an automated LLM judge (Qwen3.6-35B-A3B) against 26–82 fine-grained criteria, reporting the **all-pass rate**; with Gemma, whose all-pass rate is near zero, we report the mean fraction of criteria met.

Models and Harness. We test two models: **Qwen3.6-35B-A3B** (extended with YaRN interpolation to a 1M context window in FP8 precision) and **Gemma-4-26B-A4B-it** (Gemma Team, 2026). The tested model M solves the tasks during training and at test time; for every method, Qwen3.6-35B-A3B serves as M_U (reflection and skill update) and as the Legal Agent Bench judge. We turn thinking off across all agent, reflection, and skill-updating invocations. Agents interact with environments within the Claude Code harness with the skill directory mounted into the workspace. The model has 40 turns per task (60 for Gemma-4-26B-A4B-it), and each model samples with its default generation configuration (for Qwen3.6-35B-A3B, T=1.0, top-p 0.95, top-k 20). RRU updates the

Table 1: **Main results.** RRU outperforms the measured baselines on the three benchmarks with both models. Mean over three seeds after one epoch of updates; colored numbers denote the change relative to No skill. Bottom block: Gemma-4-26B-A4B-it (Gemma Team, 2026) solves the tasks and Qwen3.6-35B-A3B performs reflection and updates; its Legal Agent Bench cells report the mean fraction of criteria met. Table 5 reports standard deviations.

	AppWorld		SpreadsheetBench	Legal Agent Bench
Method	**TGC (%) ↑**	**SGC (%) ↑**	**Acc. (%) ↑**	**Score (%) ↑**
Qwen3.6-35B-A3B				*all-pass*
No skill	58.8	35.1	59.4	13.9
Trace2Skill (Ni et al., 2026)	62.7 +3.9	38.1 +3.0	62.6 +3.2	11.3 −2.6
SkillOpt (Yang et al., 2026)	61.4 +2.6	42.3 +7.2	66.0 +6.6	16.0 +2.1
Combee (ACE) (Li et al., 2026a)	67.4 +8.6	43.2 +8.1	56.5 −2.9	12.3 −1.6
RRU (**Ours**)	**72.3** +13.5	**53.6** +18.5	**73.1** +13.7	**19.7** +5.8
Gemma-4-26B-A4B-it				*mean criteria*
No skill	35.1	16.1	60.4	58.7
Trace2Skill (Ni et al., 2026)	43.8 +8.7	22.0 +5.9	65.3 +4.9	57.3 −1.4
SkillOpt (Yang et al., 2026)	48.0 +12.9	24.7 +8.6	62.3 +1.9	59.0 +0.3
Combee (ACE) (Li et al., 2026a)	42.4 +7.3	19.9 +3.8	60.3 −0.1	55.5 −3.2
RRU (**Ours**)	**60.1** +25.0	**36.0** +19.9	**69.4** +9.0	**60.2** +1.5

skill after every batch of b=16 tasks; the baselines use the default configurations of their released implementations (e.g., SkillOpt updates after batches of 40 tasks).

Baselines. We compare RRU with the following methods under the same harness, turn caps, and sampling parameters: (1) **No skill**: the model performing tasks without procedural instructions; (2) **Trace2Skill** (Ni et al., 2026): analyzes rollouts in parallel and hierarchically merges recurring lessons into a skill; (3) **SkillOpt** (Yang et al., 2026): iteratively refines candidate skills through execution feedback and validation-gated updates; and (4) **Combee (ACE)** (Zhang et al., 2026a; Li et al., 2026a): extends ACE and updates its skill (a playbook) by aggregating reflections over batches with MapReduce. All skill generation methods start from an empty skill and make one epoch of updates over the training split; to provide records for its first replay, RRU first collects one rollout per training task with the empty skill. At test time, the agent cannot read the record files. We report the mean over three random seeds (details in Appendix B).

5.1 Main Results

We present the main results in Table 1. With Qwen3.6-35B-A3B, RRU improves the TGC on AppWorld from 58.8% to 72.3% and the SGC from 35.1% to 53.6%, the accuracy on SpreadsheetBench from 59.4% to 73.1%, and the all-pass rate on Legal Agent Bench from 13.9% to 19.7%, and outperforms the measured baselines. Compared with the best baseline on each benchmark, it improves the TGC by 4.9% and the SGC by 10.4% on AppWorld, the accuracy by 7.1% on SpreadsheetBench, and the all-pass rate by 3.7% on Legal Agent Bench. We observe similar patterns with Gemma-4-26B-A4B-it: RRU improves the TGC on AppWorld from 35.1% to 60.1%, outperforming SkillOpt by 12.1%, and also outperforms the baselines on SpreadsheetBench (69.4% vs. 65.3%) and Legal Agent Bench (60.2% vs. 59.0%).

The skills learned by RRU capture different knowledge on each benchmark (excerpts in Appendix C). On AppWorld, they mostly describe how the application APIs behave and which data source a task refers to (e.g., phone contacts or a Venmo friend list). On SpreadsheetBench, many tasks depend on a single instruction: write the computed value unless the task asks for a formula. All three RRU skills keep this instruction with its condition, which the update without replay removes in Figure 3. SkillOpt skills also contain this instruction, but among many more instructions (196–785 lines, against 120–163 lines for RRU). On Legal Agent Bench, the RRU skills mainly list what the criteria check, such as reading each source document and using the template's defined terms.

We also observe that well-established baselines improve over the model without a skill in most settings: SkillOpt on the three benchmarks with both models, Trace2Skill and Combee on AppWorld, and Trace2Skill also on SpreadsheetBench. Their skills, however, sometimes lower accuracy: Trace2Skill and Combee fall below the model without a skill on Legal Agent Bench with both

Table 2: **Ablation of the replay stage.** The replay stage (§4.2.2) improves accuracy on both benchmarks with both models. Mean ± population standard deviation over three seeds; colored numbers denote the change relative to the variant without the replay stage, which uses the same prompts. With Gemma-4-26B-A4B-it, Qwen3.6-35B-A3B performs reflection and updates.

	AppWorld		SpreadsheetBench
	TGC (%)	**SGC (%)**	**Acc. (%)**
Qwen3.6-35B-A3B			
No skill	58.8	35.1	59.4
w/o replay stage	66.1 ± 7.8	47.9 ± 10.2	67.2 ± 5.7
RRU	**72.3** ± 2.1 +6.2	**53.6** ± 2.5 +5.7	**73.1** ± 1.9 +5.9
Gemma-4-26B-A4B-it			
No skill	35.1	16.1	60.4
w/o replay stage	53.7 ± 3.5	32.1 ± 2.6	67.9 ± 0.3
RRU	**60.1** ± 2.3 +6.4	**36.0** ± 3.4 +3.9	**69.4** ± 1.0 +1.5

Table 3: **Ablation on the representation of a record.** Compact records (§4.2) reach higher accuracy than complete rollouts and a lower token cost, each in three of the four settings. Mean over three seeds; colored numbers denote the change relative to the variant without the replay stage. SSB: SpreadsheetBench. Avg. Token Cost: tokens of the rollouts and skill updates in one training run.

	AppWorld		SpreadsheetBench	Avg. Token Cost	
Replayed content	**TGC (%)**	**SGC (%)**	**Acc. (%)**	**AppWorld**	**SSB**
Qwen3.6-35B-A3B					
w/o replay stage	66.1	47.9	67.2	40.8M	56.1M
Complete rollouts	**73.7** +7.6	**58.6** +10.7	71.5 +4.3	103.2M	180.2M
Compact records (RRU)	72.3 +6.2	53.6 +5.7	**73.1** +5.9	72.7M	125.4M
Gemma-4-26B-A4B-it					
w/o replay stage	53.7	32.1	67.9	32.2M	48.3M
Complete rollouts	55.3 +1.6	**36.6** +4.5	67.2 −0.7	68.7M	95.2M
Compact records (RRU)	**60.1** +6.4	36.0 +3.9	**69.4** +1.5	80.3M	83.9M

models, and Combee also on SpreadsheetBench. The baselines also vary more across seeds: with Qwen, the standard deviation is 8.5% for Trace2Skill in TGC on AppWorld, where one seed reaches 51.8%, below the model without a skill (58.8%), and 9.6% for Combee in accuracy on SpreadsheetBench, against 2.1% and 1.9% for RRU, whose mean does not fall below the model without a skill in any setting.

5.2 Ablations

In this section, we present two ablation studies on AppWorld and SpreadsheetBench with both models: (1) the replay stage (§4.2.2), and (2) the representation of a record (§4.2).

First, we study the effect of the replay stage. We remove it, so that U conditions only on $\mathcal{R}_k$ and S_{k-1} as in Eq. (1), and keep the prompts and the rest of RRU unchanged (Table 2). This variant is already competitive with the established baselines: its accuracy is comparable to or above that of the best baseline in each setting of Table 1 (e.g., 66.1% vs. 67.4% TGC on AppWorld and 67.2% vs. 66.0% accuracy on SpreadsheetBench with Qwen). Adding the replay stage improves the TGC on AppWorld from 66.1% to 72.3% and the accuracy on SpreadsheetBench from 67.2% to 73.1% with Qwen, and improves them by 6.4% and 1.5% with Gemma.

We also observe that the replay stage improves stability across seeds. Without the replay stage, the standard deviation with Qwen is 7.8% on AppWorld and 5.7% on SpreadsheetBench, and the worst AppWorld seed (55.4%; Table 4) falls below the model without a skill, as in Figure 2. With the replay stage, the standard deviations are 2.1% and 1.9%, the worst seed is above the mean without the replay stage, and the best seeds of the two variants differ by only 1.5% and 0.7%. This suggests that the replay stage mainly protects the weaker seeds rather than raising the best case.

Second, we study how a record represents a rollout. We replace the compact record $\rho(\tau_x)$ with the complete rollout τ_x, which is eight to nine times longer (§4.2), and keep the rest unchanged (Table 3). Compact records are higher in three of four settings: on SpreadsheetBench with both

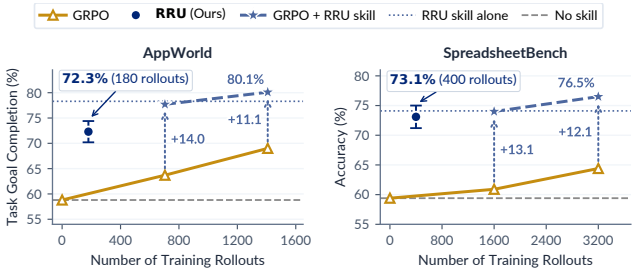


Figure 5: **Skill generation and reinforcement learning.** Without updating the model, RRU outperforms two epochs of GRPO with 7.8×–8.0× fewer rollouts, and adding its skill to each GRPO checkpoint improves it by 11.1–14.0%. Dotted line: the separate RRU skill (§5.3) alone.

models and in TGC on AppWorld with Gemma (60.1% vs. 55.3%); complete rollouts are higher on AppWorld with Qwen (73.7% vs. 72.3% TGC and 58.6% vs. 53.6% SGC). With Qwen, compact records use 30% fewer tokens; with Gemma, they use 12% fewer tokens on SpreadsheetBench and 17% more on AppWorld. Averaged over the four settings, compact records are 1.8% higher in TGC and accuracy. Compared with the variant without the replay stage, RRU uses 1.7 to 2.5 times more tokens per training run, most of them in rollouts, including the initial record collection.

5.3 Comparison with Reinforcement Learning

Reinforcement learning (RL) improves an agent from its own rollouts, by updating the model's weights instead of a skill. We study the relationship between RRU and RL in two scenarios. *Without fine-tuning access* (training is too costly or the model is available only through an API), we compare skill generation with what RL would achieve; *with fine-tuning access*, we study whether a skill adds to the gains of RL. GEPA (Agrawal et al., 2026) found that reflective prompt optimization can outperform GRPO with fewer rollouts. We train Qwen3.6-35B-A3B with GRPO (Shao et al., 2024) in the same harness (group size 8, 64 rollouts per step) for two epochs (1,408 rollouts on AppWorld and 3,200 on SpreadsheetBench), with binary task-completion and unit-test rewards (one run each).

Without fine-tuning access, RRU outperforms GRPO with fewer rollouts (Figure 5). After two epochs, GRPO reaches a TGC of 69.0% on AppWorld (from 58.8%) and an accuracy of 64.4% on SpreadsheetBench (from 59.4%). RRU reaches 72.3% and 73.1% with 180 and 400 rollouts, which are 7.8 and 8.0 times fewer. One training run of RRU costs $4.47 on AppWorld and $8.23 on SpreadsheetBench at OpenRouter list prices (Appendix B).

With fine-tuning access, a skill can be added to a fine-tuned model without further training. We add the skill of a separate RRU training run (78.3% on AppWorld and 74.1% on SpreadsheetBench on its own) to the GRPO checkpoints after each epoch. After two epochs, the skill improves the TGC on AppWorld from 69.0% to 80.1% and the accuracy on SpreadsheetBench from 64.4% to 76.5%, above both the skill alone and GRPO alone.

6 Conclusion

We presented RRU, which maintains compact records of the rollouts that provide evidence supporting each instruction and replays these rollouts during skill updates, so that an update can revise an instruction rather than remove it. Across three benchmarks and two models, RRU outperforms existing frameworks and is more stable across seeds in most settings, and its skills can be added to models fine-tuned with reinforcement learning.

Limitations and future work. An update can be grounded by giving M_U more context, as RRU does with the records that support each instruction, or by re-executing tasks to test an edit. With more context, M_U can still misread the records; re-execution is also noisy, since rollouts are sampled. We chose context because we expect stronger models to use sufficient context more reliably, and how to combine the two is an open question. On Legal Agent Bench, the judge is the same model as M_U. Future work also includes skills that are updated while an agent is in use.

AI USE STATEMENT

We used generative AI tools in the following ways. *Writing*: to polish the text and to draft parts of sections, which the authors then revised. *Retrieval and discovery*: to search for and summarize related work; we checked every cited reference against its arXiv, DOI, or official source. *Research ideation and execution*: to discuss method variants and experiment designs, to implement code, to run and monitor experiments, and to analyze results and produce figures. The authors decided the research direction, the method, and the claims, and reviewed all AI-assisted work; we take responsibility for the final content, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses publicly available benchmarks (AppWorld, SpreadsheetBench, and Legal Agent Bench) and open-weight models, and involves no human subjects or personal data. The generated legal skills are research artifacts evaluated by an automated judge and are not legal advice.

REPRODUCIBILITY STATEMENT

Section 5 and Appendix B describe the benchmarks and splits, models, harness, sampling parameters, training settings of every method, and evaluation protocol; all skill generation results are means over three seeds, with standard deviations and per-seed values of the replay ablation in Appendix A. We will release the code, the prompts, and the generated skills.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Anthropic. Equipping Agents for the Real World with Agent Skills. `https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills`, October 2025. Anthropic Engineering Blog.

Johan De Kleer. An assumption-based TMS. *Artificial intelligence*, 28(2):127–162, 1986.

Jon Doyle. A truth maintenance system. *Artificial intelligence*, 12(3):231–272, 1979.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Harvey AI. Harvey Legal Agent Benchmark (LAB): a benchmark built to evaluate and improve agent capabilities for supporting legal work. `https://github.com/harveyai/harvey-labs`, 2026. Version at commit c2488cfa2.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. SoK: Agentic skills–beyond tool use in LLM agents. *arXiv preprint arXiv:2602.20867*, 2026.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Hanchen Li, Runyuan He, Qizheng Zhang, Changxiu Ji, Qiuyang Mang, Xiaokun Chen, Lakshya A Agrawal, Wei-Liang Liao, Eric Yang, Alvin Cheung, et al. Combee: Scaling prompt learning for self-improving language model agents. *arXiv preprint arXiv:2604.04247*, 2026a.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. SpreadsheetBench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

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. *arXiv preprint arXiv:2305.16291*, 2023.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Jiayi Weng. Learning Beyond Gradients. `https://trinkle23897.github.io/learning-beyond-gradients/`, May 2026. Blog post.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026a.

Qizheng Zhang, Michael Wornow, Gerry Wan, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents. *Advances in Neural Information Processing Systems*, 38:103270–103296, 2026b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Results per Seed

Table 4 lists the three seeds that support Table 2.

Table 4: **Replay ablation per seed.** With replay, every seed exceeds the mean without replay.

		AppWorld TGC (%)			SpreadsheetBench Acc. (%)		
Model	**Replay**	Seed 1	Seed 2	Seed 3	Seed 1	Seed 2	Seed 3
Qwen3.6-35B-A3B	without	55.4	69.3	73.5	61.0	74.7	65.8
	with (RRU)	69.9	72.0	75.0	70.7	75.4	73.3
Gemma-4-26B-A4B-it	without	53.6	58.0	49.4	68.4	67.7	67.7
	with (RRU)	61.6	56.8	61.9	69.1	68.4	70.8

Main results with standard deviations. Table 5 reports the results of Table 1 with their standard deviations over the three seeds. The no-skill cells without a standard deviation come from a single evaluation.

B Experimental Details

Benchmarks and evaluation. AppWorld: 90 training tasks and the 168-task *test-normal* split; we report TGC and SGC over two attempts per test task. SpreadsheetBench: 200 training and 144 test tasks; accuracy is the fraction of attempts whose output passes the checker, over eight attempts per test task with Qwen3.6-35B-A3B and two with Gemma-4-26B-A4B-it. Legal Agent Bench: 48 training and 48 test tasks from the two memo task kinds (§5); each output is graded by the judge against its task's criteria, over three attempts per test task with Qwen3.6-35B-A3B (all-pass rate) and one with Gemma-4-26B-A4B-it (mean fraction of criteria met).

Models and serving. Qwen3.6-35B-A3B runs in FP8 with YaRN scaling to a 1M-token context window, and Gemma-4-26B-A4B-it with a 262k-token window; both are served with SGLang. Every agent, reflection, and update call uses the model's default generation configuration with thinking turned off (for Qwen3.6-35B-A3B, T=1.0, top-p 0.95, top-k 20).

Harness. The agent works in the Claude Code harness with the skill directory mounted in its workspace and a cap of 40 turns per task (60 for Gemma-4-26B-A4B-it). An episode ends at its first graded submission, at the turn cap, or at a wall-clock limit. At test time, the record files are not accessible to the agent.

RRU. RRU starts from an empty skill, collects one rollout per training task with the empty skill to provide records for the first replay, and then makes one epoch of updates with batches of b=16 tasks. Each record is stored as a file in the skill folder, and each instruction in `SKILL.md` cites the records in its support set. The variant without the replay stage uses the same prompts and batches.

Implementation of the skill folder. The skill is a folder in the format of Agent Skills (Anthropic, 2025):

```
skill/
  SKILL.md              instructions; each cites its support set as a list of task ids
  references/<file>.md  optional files that SKILL.md points to
  assets/raw/<task>.md  the record of a training task (task, actions, outcome, reflection);
                        a later attempt at the same task is appended to the same file
```

Baselines. Trace2Skill, SkillOpt, and Combee run with the default configurations of their released implementations in the same harness, with the same models, turn caps, and sampling parameters, starting from an empty skill and making one epoch over the training split (e.g., SkillOpt updates after batches of 40 tasks).

Table 5: **Main results with standard deviations.** Mean ± standard deviation over three seeds (population, as in Table 2) for the results in Table 1; other settings as in Table 1.

	AppWorld		SpreadsheetBench	Legal Agent Bench
Method	**TGC (%) ↑**	**SGC (%) ↑**	**Acc. (%) ↑**	**Score (%) ↑**
Qwen3.6-35B-A3B				*all-pass*
No skill	58.8±2.8	35.1±3.7	59.4	13.9
Trace2Skill (Ni et al., 2026)	62.7±8.5	38.1±10.4	62.6±3.0	11.3±0.9
SkillOpt (Yang et al., 2026)	61.4±2.8	42.3±4.0	66.0±5.6	16.0±2.6
Combee (ACE) (Li et al., 2026a)	67.4±4.6	43.2±7.9	56.5±9.6	12.3±1.7
RRU (**Ours**)	**72.3**±2.1	**53.6**±2.5	**73.1**±1.9	**19.7**±2.1
Gemma-4-26B-A4B-it				*mean criteria*
No skill	35.1	16.1	60.4	58.7±1.8
Trace2Skill (Ni et al., 2026)	43.8±3.4	22.0±0.4	65.3±6.2	57.3±4.2
SkillOpt (Yang et al., 2026)	48.0±5.2	24.7±4.7	62.3±0.3	59.0±1.7
Combee (ACE) (Li et al., 2026a)	42.4±3.4	19.9±0.8	60.3±0.9	55.5±1.1
RRU (**Ours**)	**60.1**±2.3	**36.0**±3.4	**69.4**±1.0	**60.2**±2.0

Reinforcement learning. We train Qwen3.6-35B-A3B with GRPO in the same harness: eight tasks per step with eight rollouts each (64 rollouts per step), binary rewards (task completion on AppWorld, unit-test pass on SpreadsheetBench), learning rate 10^{-6}, no KL penalty, token-level truncated importance sampling (clipped at 2.0), sampling temperature 1.0, a 100k-token context, and two epochs over the training split. Costs in §5.3 use the OpenRouter list prices for Qwen3.6-35B-A3B, $0.05 and $0.70 per million input and output tokens.

C Excerpts of Generated Skills

We show one instruction from a skill generated by RRU with Qwen3.6-35B-A3B on each benchmark; each skill is the one evaluated for the corresponding seed in Table 1. Excerpts are verbatim Markdown from `SKILL.md`; [...] marks omitted text, and a ↪ list names the training tasks whose records support the instruction. An instruction can span several bullets, and its ↪ list follows the part of the instruction it was attached to.

AppWorld (seed 1; 65 lines). The instruction separates tasks that require an answer from tasks that only require an action.

```
### Complete_task: action vs answer — check before submitting

**Before calling `apis.supervisor.complete_task()`, ask: "Does the task ask me to *tell* something (answer), or *do* something (no answer)?"**

- **Action-only tasks** (no answer): If the task only asks you to perform an action (like, remove, transfer, rate, send a text/message, add a comment, compress, delete), call `apis.supervisor.complete_task()` with **no arguments**. Sending a non-null `answer` string will cause the grader to reject it (it expects `'null'`). This applies even when you believe the task "also needs a report" — if the task verb is purely imperative ("Rate all songs...", "Add a comment...", "Send a text..."), no answer is needed. ↪ 692c77d_1, afc0fce_2, 29caf6f_2, ccb4494_3, 34d9492_1, 6104387_1, 229360a_1, 771d8fc_2, 771d8fc_3, 07b42fd_3, 34d9492_3, e3d6c94_1, e3d6c94_3, 7d7fbf6_2
- **Query tasks** (answer needed): If the task asks for specific information ("How much...?", "What is...?", "Which...?"), provide the answer via `complete_task(answer=<value>)`.
- **Action + answer tasks**: If the task asks both for an action AND a numeric/factual answer, perform the action AND provide the answer.
[...]
```

SpreadsheetBench (seed 3; 120 lines). The instruction discussed in §5.1: compute values in Python unless the task asks for a formula, with the condition under which a formula is written.

```
### Principle 1: Compute values in Python, write them directly — never write formula strings where the grader reads `.value`

When the task asks you to **populate cells with results** (e.g., times, numbers, text), compute the correct values in Python and write them directly to `cell.value`. Do NOT write formula strings (e.g., `cell.value = "=IF(...)"`). openpyxl stores formulas as raw
```

```
     text; when the grader reads `cell.value`, it gets the formula string, not the computed
     result.

**How to tell which approach to use:**
- If the expected value is a concrete computed value (e.g., `datetime.time`, a number, a text
     string, or blank `None`), compute it in Python and write the value directly.
- **If the task explicitly says "write a formula", "modify the formula", "create a formula",
     or similar formula-specific language**, or **if the input workbook already contains SUM
     formulas in TOTAL/summary rows**, you may write the formula string — but ONLY if the
     formula is a simple row-local computation using standard single-cell functions (IF, SUM,
     COUNTIF, simple INDEX/MATCH without array coercion, etc.). [...]
[...]
↪ 49667, 58484, 42515, 524-31, 290-1, 48982, 54667, 44266, 6698, 55421, 54085, 49237, 57558,
     37554, 53117, 9726, 58032, 38969, 56419, 41-47, 56921, 34033, 46240, 50088, 37456,
     41420, 51-12
```

Legal Agent Bench (seed 2; 107 lines). The instruction asks the agent to write proposed terms as contract clause language in the conventions of the source template.

```
### 7. Draft contract clause language must use the industry-specific conventions from the
     baseline/template

[...]
**What to do:**
- When the task asks for "draft contract clause language" or "compromise clause language,"
     include standalone, numbered clause text for each proposed term using the defined-term
     conventions from the baseline/template document (e.g., YC SAFE conventions for SAFE
     negotiations, standard SPA sections for M&A).
- Include specific defined terms (e.g., "Cap Price," "Discount Price," "Equity Financing")
     and industry-standard mechanic language (e.g., "lower of" dual-trigger conversion for
     discount rates).
[...]
- Structure the proposed language like actual contract text: numbered sections, defined terms
     in **bold** or "defined" format, operative verbs ("shall," "will").
- After writing prose rationale, add a separate "Proposed Clause Language" subsection for
     each term with the actual draft text. ↪ harvey_financing__safe-instrument-term-
     negotiation, harvey_commercial-vendor-customer__master-services-agreement-term-
     negotiation__scenario-01, harvey_banking__isda-master-pack-term-negotiation__scenario
     -02, harvey_commercial-channel-partnerships__strategic-partnership-co-sell-term-
     negotiation__scenario-02, harvey_commercial-vendor-customer__master-services-agreement-
     term-negotiation__scenario-02, harvey_banking__account-control-agreement-term-
     negotiation
```

D EXCERPTS OF THE PROMPTS

Each stage of RRU is a session of M_U guided by a Markdown prompt. We show abridged excerpts; [...] marks omitted text. Parts marked with † refer to records or to earlier attempts and are removed in the variant without the replay stage (§5.2).

Reflect: the reflection on a rollout (§4.2.1). After a rollout is graded, its session continues with M_U, which writes a reflection on the attempt in fixed sections; the reflection becomes part of the rollout's record.

```
Write a report on your attempt. The skill's author reads this report to improve the skill.
     [...]
## What happened
- Task: In one sentence, state what was asked and which cells or files were modified.
- Key actions: 3 to 6 main steps taken in order, each with the command run or file edited
     [...].
- Verdict against earlier attempts: <likely chance | pass -> pass | fail -> pass | fail ->
     fail | pass -> fail> [...] (†)
- Grader output: Quote the grader's detail verbatim as the grader wrote it (or write "passed
     ").
## Cause
State the primary cause in one sentence based directly on your execution trace. [...] If your
     explanation is an inference not directly proven by a step in your trace, label it "
     guess".
## The skill
For each instruction you followed or ignored: quote the instruction verbatim, state whether
     it was followed or not followed, and describe its effect as shown in your trace. [...]
## Environment and tool notes (optional)
Only facts directly revealed by command outputs [...]. One line each, with the command that
     showed it.
```

Reflect: recurring patterns across the batch (§4.2.1). M_U reads the reflections of the batch and writes $\mathcal{R}_k$ as a list of findings.

```
A finding is ONE cause shared by two or more tasks, and the cause is a mechanism: something
    about the environment, the tools, or the grader that would produce the same failure in a
     task with a different business rule [...].
- Observed: Quote concrete facts directly from the reports [...].
- Cause: The shared root cause across these tasks. State what evidence supports it and what
    would disprove it [...].
- Suggestion (optional): What the skill should add, modify, or remove, and how a solver can
    check this BEFORE submitting. [...]
- Supporting task IDs: ↪ <id>, <id> (†)
[...]
Seen once. An observation from a single attempt is not a finding. [...]
```

Replay and Update (§4.2.2, §4.2.3). For each finding, M_U quotes the existing instructions it concerns, checks their records, and states a verdict before it edits `SKILL.md`.

```
finding <n>: <what general mechanism or pattern this represents>
  existing rule: "<quote exact instruction from current SKILL.md>" -- or: none
  records checked: <the ids that instruction cites and which you opened>, <this batch's ids
    you opened> (†)
  verdict: confirms rule | refines boundary | breaks rule | new pattern -- <one sentence:
    what the evidence shows>
  [...]
You may not edit an instruction unless you quote it above; you may not cite a record unless
    you opened and checked it (†); a verdict of "new pattern" requires clear evidence from
    two or more tasks that no existing instruction covers.
[...]
- Unify overlapping and conflicting rules: When two instructions say essentially the same
    thing, or contradict each other, combine them into one clear, unifying rule and keep
    both sets of citation marks (↪ <task id>, <task id>) (†). [...]
- Confirmations and boundary cases: If a finding simply confirms an existing instruction,
    just add its supporting task ID to the citations: ↪ <task id> (†). [...]
[...]
Rejection is the default, healthy outcome for individual tasks. [...]
```

E ACCURACY AFTER EACH UPDATE ON SPREADSHEETBENCH

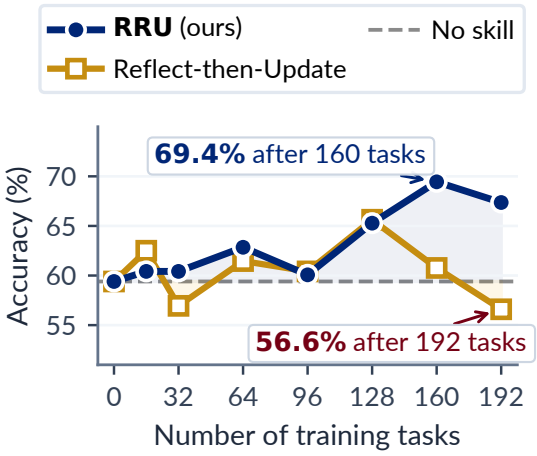


Figure 6: **Accuracy after each update on SpreadsheetBench** (Qwen3.6-35B-A3B, one seed; the counterpart of Figure 2). Test accuracy after each update (batches of 16 tasks). Without replay, accuracy oscillates and falls to 56.6% after 192 training tasks, below the model without a skill (59.4%); with replay it peaks at 69.4% and ends at 67.4%. This curve is a separate single-seed run, so its final skill differs from those in Table 2.