CTRLSKILL: A CONTROL-THEORETIC VIEW OF CONTROLLABLE SKILL SELF-EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents can distill reusable skills from execution trajectories, yet accumulating experience does not necessarily yield reliable performance gains. Success on local validation is insufficient to justify persistent skill adoption, failures with mixed causes can lead to spurious generalization, and tool execution errors may be misattributed to reasoning deficiencies. We introduce CtrlSkill, a Control-Theoretic View of Controllable Skill Self-Evolution that organizes updates to an external skill policy as an evidence-driven feedback loop. CtrlSkill first constructs structured evidence to distinguish intended actions, tool execution, and observed outcomes. It then converts shared evidence gaps into candidate skills with explicit applicability conditions and verification requirements. Finally, it compares the active and candidate policies on subsequent matching tasks, controlling whether updates take effect through acceptance, deferral, or rejection. By decoupling candidate generation from policy commitment, CtrlSkill subjects skill updates to explicit constraints on supporting evidence, edit scope, and validation feedback. Across these settings, CtrlSkill raises the overall average score over baselines by up to 23%.

1 INTRODUCTION

Large language model (LLM) agents increasingly move beyond answering individual questions to invoking tools, changing environment states, and producing task artifacts. Alongside model parameters, readable and editable external skills have become an important medium for accumulating execution experience. Reflexion (Shinn et al., 2023) stores verbal feedback in memory, ExpeL (Zhao et al., 2024)extracts reusable rules from task experience, and Voyager organizes successful programs into a skill library (Wang et al., 2023). These approaches demonstrate that agents can improve subsequent behavior by updating external knowledge and execution strategies while keeping model parameters fixed.

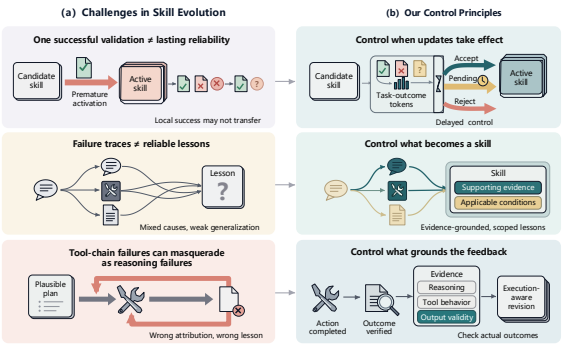

Figure 1: Reliability challenges in skill self-evolution and the corresponding control principles of CtrlSkill. Delayed admission controls when updates take effect; evidence-grounded, conditional editing controls what becomes a skill; and execution-aware observation controls what grounds the feedback.

Recent work further studies the automatic generation and revision of skills. Trace2Skill (Ni et al., 2026) distills transferable lessons from execution trajectories, while SkillCAT (Chen et al., 2026) improves skill evolution through contrastive assessment and candidate replay. SkillDAG (Bai et al.), SKILLFOUNDRY (Shen et al., 2026), and Library Drift (Zhang et al., 2026c) investigate skill organization, construction, and library-level maintenance, respectively. However, generating plausible skill text and establishing sufficient evidence for its use on future tasks are distinct problems. We focus on the latter: as trajectories arrive online, how should an agent control the evidence, content, and activation of skill updates?

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.

Figure 1 illustrates three aspects of this problem. First, a successful validation does not establish lasting reliability: a candidate may repair its source example but fail on subsequent tasks. Second, failure traces are not necessarily reliable lessons: the same outcome can arise from different causes, and direct summarization can turn incidental factors into general rules. Third, tool-chain failures can be misattributed to reasoning errors. A plausible plan, a completed action, and an accepted outcome are different states; reflection at the language level alone may therefore target the wrong part of the execution process.

We introduce CtrlSkill, which formulates skill self-evolution as an outer supervisory control process. The agent execution system under the current skill policy acts as the plant; an observer builds an evidence state from trajectories, tool returns, and artifact checks; and a controller proposes scoped edits and uses subsequent feedback to decide whether they should be committed. A candidate therefore remains a hypothesis until admission and does not alter the active policy during generation. As illustrated in the right column of Figure 1, delayed admission controls *when updates take effect*, conditional skill editing controls *what becomes a skill* through supporting evidence and applicability conditions, and execution-aware observation controls *what grounds the feedback* by separating reasoning, tool behavior, and output validity. Together, these principles form the observation–editing–admission loop shown in Figure 2.

Our contribution is a control-inspired framework for controllable skill self-evolution that connects structured execution evidence, conditional repair hypotheses, and delayed admission through traceable candidate validation and policy commitment. The experimental protocol evaluates frozen skill policies, candidate updates, and reuse across models.

2 Related Work

2.1 Agent Self-Evolution

Agent self-evolution uses feedback from task execution and environmental interaction to improve subsequent behavior. Existing approaches operate primarily at the levels of memory, strategy, and implementation. Reflexion (Shinn et al., 2023) converts execution feedback into verbal reflections, while ExpeL (Zhao et al., 2024) extracts reusable rules from experience, enabling agents to accumulate knowledge without parameter updates. AgentEvolver (Zhai et al., 2025), Agent0 (Xia et al., 2025), and Agents of Change (Belle et al., 2025) investigate autonomous task generation, co-evolution, and multi-role revision, expanding the tasks and feedback available for evolution. The Darwin Gödel Machine directly modifies agent code, making the execution system itself an object of evolution (Zhang et al., 2026b). Across these approaches, execution feedback is converted into persistent updates that influence future behavior.

Skill self-evolution is an important direction within this broader space: it turns local execution experience into reusable capabilities. Skills package procedures, applicability conditions, and tool-use knowledge into independently editable units, supporting targeted revision and reuse across tasks. Voyager (Wang et al., 2023) accumulates capabilities through a programmatic skill library; subsequent work uses skills for execution-time intervention and studies the joint evolution of task skills and evolution mechanisms. Liu et al. (2026a) represent skills as executable programs, enabling LLM agents to reuse operational knowledge and intervene in task execution through structured skill programs. Wang et al. (2026) propose a two-timescale meta-skill evolution framework that recursively improves the skill-generation and evolution mechanism while updating task-level skills. Skills thus serve both as a store of experience and as an interface between feedback and future execution. We study how evidence constraints and feedback-based validation can control the formation and activation of these updates.

2.2 Skill Self-Evolution

Trajectory-driven skill evolution extracts reusable rules from execution experience. Trace2Skill (Ni et al., 2026) emphasizes distilling trajectory-local lessons into transferable skills, and SkillCAT (Chen et al., 2026) combines contrastive assessment with candidate replay to improve skill updates . MemSkill (Zhang et al., 2026a)and SkillOpt (Yang et al., 2026)further explore the continual revision of memory skills and skill documents. These approaches provide foundations for experience distil-

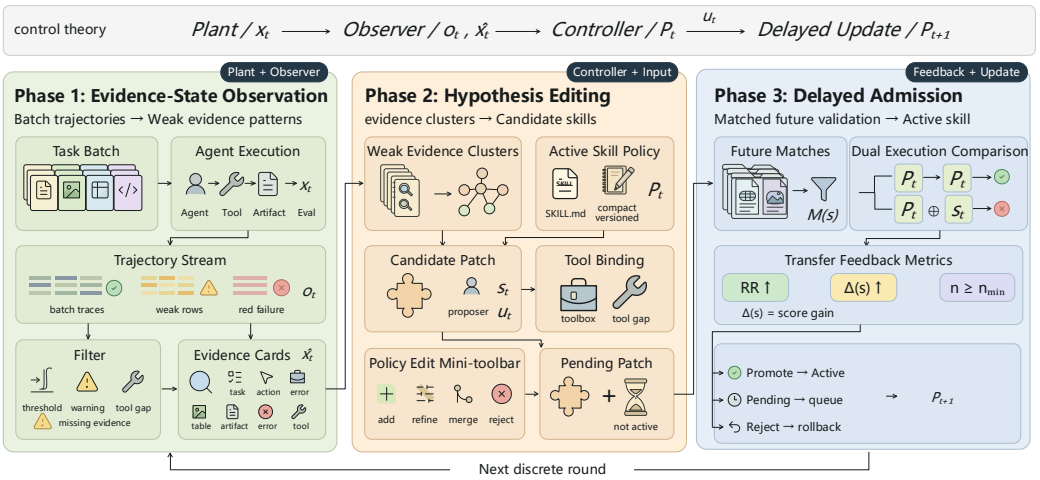

Figure 2: **The CtrlSkill feedback loop.** At round t, the system observes execution evidence O_t and maintains a filtered state $\hat{x}_t$ from the task state x_t. The controller uses $\hat{x}_t$ to propose an edit action u_t for the active policy P_t. After validation on matched tasks, the policy is delayed-updated to P_{t+1}, separating local failure observation from skill activation.

lation and feedback-based selection. CtrlSkill uses validation on subsequent matching tasks as the basis for delayed admission, separating candidate generation from commitment to the active policy.

As skill libraries grow, their organization and lifecycle management also affect long-term utility. SkillDAG (Bai et al.), SKILLFOUNDRY (Shen et al., 2026), Library Drift (Zhang et al., 2026c), and SkillsVote (Liu et al., 2026b) study skill relationships, construction from heterogeneous resources, library-level drift, and lifecycle governance, respectively. CtrlSkill focuses on the admission of candidate edits: structured evidence limits the repair scope, tool and artifact states ground the feedback, and records link each candidate to its target versions and validation outcomes. This mechanism is compatible with retrieval, graph-based organization, and library-level governance.

3 Method

3.1 Problem Formulation and Control-Theoretic View

CtrlSkill separates skill generation, trial execution, and activation. As shown in Figure 2, the Plant and Observer construct the evidence state, the Controller proposes candidate edits, and Delayed Update determines whether they affect the active policy.

Consider a tool-augmented agent π_θ with fixed parameters θ and external skill policy P_t. For task z in environment $\mathcal{E}_z$, execution produces trajectory τ_z, outcome a_z, and score:

$$(\tau_z, a_z) \sim \mathrm{Rollout}(\pi_\theta, P_t, z, \mathcal{E}_z), \qquad r_z = R(z, a_z) \in [0, 1]. \tag{1}$$

The outcome may be an answer, program, document, data artifact, or environment change. Skill evolution seeks to improve future-task performance without updating θ:

$$J(P) = \mathbb{E}_{z\sim\mathcal{D}}\left[\mathbb{E}_{(\tau_z,a_z)\sim\mathrm{Rollout}(\pi_\theta,P,z,\mathcal{E}_z)}\left[R(z, a_z)\right]\right]. \tag{2}$$

Because the task distribution $\mathcal{D}$ is unknown, the utility of a candidate policy must be established through execution feedback. Therefore, generating a skill only proposes a policy modification; it does not authorize that modification to take effect.

Inspired by feedback control (Åström & Murray, 2008), we treat the skill-conditioned agent as the plant. The observer maps execution feedback to an evidence state, the controller proposes an edit, and the admission mechanism determines when the edit changes the active policy. Let x_t, o_t, and $\hat{x}_t$

denote the execution state, observation, and observer output:

$$\begin{aligned} o_t &= h(x_t, \nu_t), \qquad \hat{x}_t = \Phi(o_t, \mathcal{H}_t), \\ (s_t, u_t) &= C(\hat{x}_t, P_t), \quad P_{t+1} = U(P_t, \mathcal{Q}_t, \mathcal{F}_t). \end{aligned} \tag{3}$$

Here, ν_t is observation noise, $\mathcal{H}_t$ is historical evidence, $\mathcal{Q}_t$ is the pending-edit set, and $\mathcal{F}_t$ is the available validation feedback.

The loop answers three questions: what diagnosis the trajectory supports, which policy modification the evidence justifies, and whether subsequent validation is sufficient. Thus, observation constrains the basis of feedback, editing constrains skill content, and admission constrains activation. Detailed analysis are provided in Appendix A.5.

3.2 Evidence-State Observation

Task failure indicates that an outcome did not meet the requirements, but does not directly identify which behavior should change. We therefore convert each execution trajectory into a structured evidence card:

$$e_z = (g_z, G_z, V_z^{\mathrm{req}}, V_z^{\mathrm{miss}}, b_z, f_z), \qquad G_z = (V_z, E_z). \tag{4}$$

Here, g_z specifies the task goal and acceptance requirements; V_z and E_z contain evidence units and their relationships; V_z^{req} and V_z^{miss} denote required and missing evidence; and b_z and f_z record behavioral cues and external feedback. Evidence nodes cover input grounding, rule interpretation, operation construction, tool execution, outcome state, and acceptance checks.

Let $\mathrm{support}(v, o_z)$ indicate whether observation o_z supports a required evidence item v. The missing-evidence set is

$$V_z^{\mathrm{miss}} = \{v \in V_z^{\mathrm{req}} : \mathrm{support}(v, o_z) = 0\}. \tag{5}$$

This representation distinguishes intended operations, actual execution, and goal attainment, corresponding to reasoning, tool behavior, and output validity in the bottom row of Figure 1. Generated code need not have been executed; a successful tool return need not imply a correct target update; and an existing file need not satisfy the acceptance criteria. Tool calls and artifact checks therefore contribute directly to the evidence state.

Evidence cards also preserve differences in evidential strength. Tool returns, environment checks, and external evaluations can substantiate the corresponding state, whereas tool names and execution claims in an answer are only behavioral cues. When observations cannot distinguish alternative causes, the system retains the evidence gap and requires further checks rather than recording conjecture as fact.

We prioritize trajectories with low scores, incomplete evidence chains, or explicit repair status:

$$w_z = \mathbf{1}[r_z < \rho \ \vee \ c_z = 0 \ \vee \ q_z \in \{\text{revise}, \text{explore}, \text{reject}\}], \tag{6}$$

where c_z denotes evidence-chain completeness and q_z is the evidence-card status. For a task batch $\mathcal{B}_t$, the weak-trajectory set is

$$\mathcal{W}_t = \{z \in \mathcal{B}_t : w_z = 1\}. \tag{7}$$

This Filter step in Figure 2 covers both failed tasks and apparently successful tasks with incomplete execution evidence.

Grouping experience by task wording or object identifiers can preserve surface similarities while overlooking shared execution gaps. We instead extract a structural signature:

$$\begin{aligned} \sigma(e_z) = \sigma_{\mathrm{role}}(e_z) \cup \sigma_{\mathrm{relation}}(e_z) \\ \cup \sigma_{\mathrm{operation}}(e_z) \cup \sigma_{\mathrm{gap}}(e_z). \end{aligned} \tag{8}$$

These sets encode abstract evidence roles, relations, operation types, and gap patterns. Detailed evidence is retained for diagnosis, while structural signatures support clustering and routing. For an existing cluster C_k with signature S_k, similarity and assignment are given by

$$\mathrm{sim}(e_z, C_k) = \frac{|\sigma(e_z) \cap S_k|}{|\sigma(e_z) \cup S_k|}, \qquad k^\star = \arg\max_k \mathrm{sim}(e_z, C_k). \tag{9}$$

If the maximum similarity reaches η, the sample joins $C_{k^\star}$; otherwise, a new cluster is created when capacity permits. Cluster signatures accumulate as

$$S_k \leftarrow S_k \cup \sigma(e_z). \tag{10}$$

Only clusters satisfying support and distinctiveness requirements proceed to candidate generation. The observer output $\hat{x}_t$ thus includes evidence cards and their structural organization, corresponding to Evidence Cards and Weak Evidence Clusters in Figure 2. It provides traceable grounds for repair hypotheses, without presupposing their utility.

3.3 Hypothesis Editing

For a sufficiently supported evidence cluster C_k, the proposer reads representative trajectories and the current policy:

$$s_t = \mathrm{Propose}(C_k, P_t). \tag{11}$$

A candidate's content can be decomposed semantically as

$$s_t = (A_t, R_t, O_t, T_t, V_t, X_t), \tag{12}$$

where A_t specifies applicability conditions, R_t evidence requirements, O_t operational constraints, T_t required tool capabilities, V_t verification criteria, and X_t exclusions. This decomposition does not prescribe a storage format.

Each candidate is a scoped repair hypothesis: under specified conditions, use the required evidence to perform an operation and check whether its outcome meets the acceptance criteria. This corresponds to Supporting evidence and Applicable conditions in the middle row of Figure 1. Supporting evidence motivates the edit, applicability conditions delimit its use, and subsequent execution determines its utility.

Tool capabilities and outcome verification jointly constrain the operation:

$$R_t \xrightarrow[T_t]{O_t} \text{outcome state} \xrightarrow{V_t} \text{acceptance feedback}. \tag{13}$$

This relation underlies Tool Binding in Figure 2. A binding specifies the capability needed to perform an operation and its verifiable effect. A concrete library, command, or resource is an instance of that capability in the current environment. Candidates can therefore express tool gaps and verification requirements without treating a tool name from a source example as a universal rule.

The control input specifies how a candidate changes the existing policy:

$$u_t = (a_t, I_t), \qquad a_t \in \{\text{add}, \text{refine}, \text{merge}, \text{replace}, \text{reject}\}, \tag{14}$$

where I_t identifies the target skills. In this editing context, a_t denotes an edit action, distinct from the task outcome a_z in Equation (1). Using the figure's notation,

$$P_t \oplus s_t \triangleq \mathrm{Edit}(P_t, s_t; u_t). \tag{15}$$

The operator $\oplus$ applies the edit specified by u_t, rather than simply appending text. Representing the active policy as a set of skills and identifying target skills with their identifiers gives

$$P_t \oplus s_t = \begin{cases} P_t \cup \{s_t\}, & a_t = \text{add}, \\ (P_t \setminus I_t) \cup \{s_t\}, & a_t \in \{\text{refine}, \text{merge}, \text{replace}\}, \\ P_t, & a_t = \text{reject}. \end{cases} \tag{16}$$

The toolbar in Figure 2 shows add, refine, merge, and reject; replace denotes the full-replacement operation also supported by the implementation. Refinement, merging, and replacement require explicit targets and are applied in the trial policy. Prior versions are retained with the validation record; a rejected candidate's trial branch is not committed.

Non-rejected candidates enter the pending set:

$$\mathcal{Q}_t \leftarrow \mathcal{Q}_t \cup \{(s_t, u_t)\}. \tag{17}$$

At this point, $P_t \oplus s_t$ is used only in the trial branch. Candidate provenance, target versions, and validation records accompany the edit, while the active policy remains unchanged.

3.4 DELAYED ADMISSION

Once generated, a candidate skill is not immediately incorporated into the active policy. Instead, it is placed in a pending queue. On subsequent tasks that match the candidate's target scope, the system performs paired evaluations of the current active policy and an edited policy augmented with the candidate. This process determines whether the candidate transfers reliably to unseen instances.

For a candidate skill s, let $b(s)$ denote its generation batch, and let $m(z, s)$ indicate whether task z matches the candidate's scope. The validation set accumulated up to batch t is

$$M_t(s) = \bigcup_{j=b(s)+1}^{t} \{z \in \mathcal{B}_j : m(z, s) = 1\}, \qquad n = |M_t(s)|. \tag{18}$$

Task–candidate matching is determined from the baseline routing before the edited policy is executed. Held-out test instances are never used for candidate generation or admission. At validation batch j, the system first evaluates the current active policy P_j and then applies the pending edit to obtain a trial policy P_j^s. For each eligible $z \in M_t(s)$, it compares the two policies under the same task conditions:

$$\begin{aligned} r_z^0 &= R\big(z; P_{j(z)}\big), \\ P_{j(z)}^s &= \mathrm{ApplyEdit}\big(P_{j(z)}, s, u_s\big), \\ r_z^s &= R\Big(z; P_{j(z)}^s\Big), \\ d_z(s) &= r_z^s - r_z^0. \end{aligned} \tag{19}$$

Here, $P_{j(z)}$ contains only edits accepted in earlier validation rounds. For refine, merge, and replace actions, the compatible target patches are removed before the candidate is inserted into the trial policy. The trial policy is used only for validation; it is committed only after the accumulated numerical gates are satisfied. During edited-policy validation, baseline routing and evidence-card construction are reused, while answer generation and rubric evaluation are recomputed under the trial policy. The candidate's average gain is defined as

$$\Delta(s) = \frac{1}{n} \sum_{z \in M_t(s)} d_z(s). \tag{20}$$

Let the baseline-failure count be

$$F_t(s) = \sum_{z \in M_t(s)} \mathbf{1}\left[r_z^0 < \rho\right]. \tag{21}$$

We measure transfer using the repair rate among baseline-failed tasks:

$$\mathrm{RR}(s) = \frac{\sum_{z \in M_t(s)} \mathbf{1}[r_z^0 < \rho \wedge d_z(s) > 0]}{F_t(s)}. \tag{22}$$

Here, $F_t(s)$ measures the available failure evidence and $\mathrm{RR}(s)$ measures the fraction of baseline failures that improve under the trial policy. The repair rate is evaluated when $F_t(s) > 0$.

A candidate passes the admission gate only when its evidence volume, baseline-failure evidence, repair rate, and average gain jointly satisfy the required criteria:

$$G_t(s) = \mathbf{1}[n \geq n_{\min} \wedge F_t(s) \geq f_{\min} \wedge \Delta(s) \geq \delta \wedge \mathrm{RR}(s) \geq \alpha]. \tag{23}$$

Here, $n_{\min}$ and $f_{\min}$ are the minimum numbers of matching turns and baseline failures. The parameters δ and α control mean gain and repair coverage, respectively. Let $H_t(s)$ denote the waiting budget consumed by the candidate. Its admission state is

$$D_t(s) = \begin{cases} \text{accept}, & G_t(s) = 1, \\ \text{reject}, & H_t(s) \geq H_{\max} \wedge G_t(s) = 0, \\ \text{pending}, & \text{otherwise}. \end{cases} \tag{24}$$

Table 1: Main results on SpreadsheetBench-Verified, full SpreadsheetBench, and WikiTableQuestions. Vrf is held-out SpreadsheetBench-Verified accuracy; Soft and Hard are measured on full SpreadsheetBench; WikiTQ tests OOD generalization.

Skill User	Setting	Method	SpreadsheetBench			OOD	Overall
			Vrf	Soft	Hard	WikiTQ	Avg
Skill Author: GPT-5.5							
GPT-5.5	Deepening	Human-Written	33.0%	32.1%	17.8%	42.5%	31.3%
		Trace2Skill + Comb.	38.0%	27.1%	16.9%	44.5%	31.6%
		CtrlSkill	**64.5%**	**49.9%**	**30.0%**	**73.0%**	**54.3%**
	Creation	Parametric	41.0%	19.6%	10.7%	51.5%	30.7%
		Trace2Skill + Comb.	35.5%	23.8%	12.0%	47.5%	29.7%
		CtrlSkill	**55.0%**	**32.2%**	**16.6%**	**67.0%**	**42.7%**
Skill Author: DeepSeek-V4-Flash							
DeepSeek-V4-Flash	Deepening	Human-Written	69.5%	23.8%	28.6%	74.0%	48.9%
		Trace2Skill + Comb.	81.5%	44.9%	28.6%	81.5%	59.1%
		CtrlSkill	**94.0%**	**46.1%**	**31.1%**	**89.0%**	**65.0%**
	Creation	Parametric	72.0%	23.1%	12.9%	80.0%	47.0%
		Trace2Skill + Comb.	80.0%	42.0%	18.8%	82.5%	55.8%
		CtrlSkill	**89.5%**	**50.8%**	**25.9%**	**87.0%**	**63.3%**

The active policy is updated only when the candidate is accepted:

$$P_{t+1} = \begin{cases} P_t \oplus s, & D_t(s) = \text{accept}, \\ P_t, & D_t(s) \in \{\text{pending}, \text{reject}\}. \end{cases} \tag{25}$$

The pending queue retains undecided candidates and receives newly generated edits from the current batch:

$$\mathcal{Q}_{t+1} = \{(s, u) \in \mathcal{Q}_t : D_t(s) = \text{pending}\} \cup \mathcal{Q}_t^{\text{new}}. \tag{26}$$

Candidates generated near the end of training receive one final train-only validation pass after the last batch, excluding their source turns. If no eligible validation turns remain, the candidate is rejected without changing the active policy. Accordingly, Promote, Pending, and Reject in the figure correspond to accepting a candidate, accumulating additional evidence, and terminating the candidate, respectively. Rollback means discarding a trial edit that fails the gate while preserving the original active policy.

4 Experiments

4.1 Experimental Setup

We use SpreadsheetBench (Ma et al., 2024) as the primary benchmark, with 200 tasks for skill evolution and a disjoint set of 200 tasks for testing. Following Trace2Skill (Ni et al., 2026) and SkillCAT (Chen et al., 2026), we distinguish *Deepening*, initialized with human-written skills, from *Creation*, without human-written skill initialization, and compare Human-Written, Parametric, and Trace2Skill + Comb. in the corresponding settings. The primary models are GPT-5.5 and DeepSeek-V4-Flash, and task accuracy measures the performance of frozen skill policies. We conducted an evolution of cross-model transfer capabilities on Kimi K3. Out-of-domain evaluation uses a fixed random subset of 200 WikiTQ (Pasupat & Liang, 2015) test examples. DocVQA Mathew et al. (2021) uses 50 examples for evolution and 100 disjoint examples for evaluation. All test sets are excluded from candidate generation, admission, and threshold selection.

Table 2: Control-oriented ablation results for CtrlSkill under GPT-5.5 Deepening. Gen., Prom., and Rej. denote generated, promoted, and rejected candidate experiences, respectively.

Method	Vrf. (%)	Gen.	Prom.	Rej.
CtrlSkill (full)	**64.5**	**6**	**3**	3
w/o Admission	45.0 ↓ 19.5	10	8	1
w/o Evidence Acc.	47.5 ↓ 17.0	15	2	13
w/o Tool Constraints	46.0 ↓ 18.5	10	4	6

Table 3: Cross-model experience comparison on SpreadsheetBench. DS-V4F denotes DeepSeek-V4-Flash.

Author / User	Setting	Vrf. (%)
GPT-5.5 / Kimi K3	No-Skill	14.0
	Human-Written	28.0 ↑ 14.0
	Trace2Skill + Comb	20.5 ↑ 6.5
	CtrlSkill (Ours)	**45.0** ↑ 31.0
DS-V4F / Kimi K3	No-Skill	14.0
	Trace2Skill + Comb	19.0 ↑ 5.0
	CtrlSkill (Ours)	**33.0** ↑ 19.0

4.2 Implementation Details

We implement online skill evolution on the Trace2Skill execution framework, keeping model parameters fixed and processing batches of 10 tasks to collect trajectories, construct evidence cards, and propose edits. Candidates are evaluated on subsequent matching turns by comparing the current active policy with a trial policy obtained by applying the candidate edit. Validation reuses baseline routing and evidence-card construction, and recomputes answer generation and rubric evaluation under the trial policy. Admission requires at least 3 matching turns and 2 baseline failures, a repair rate of at least 0.25, and a mean gain of at least 0.02. Candidates remain pending while evidence is insufficient and are given a final train-only validation pass after the last training batch. Accepted skills that have not been superseded are frozen for testing.

4.3 Main Results

Table 1 summarizes the results on SpreadsheetBench and WikiTQ. CtrlSkill achieves the best result within every completed method block across both skill-author models and both Deepening and Creation settings. Its advantage is consistent across Vrf, Soft, Hard, WikiTQ, and the overall average.

Under GPT-5.5 Deepening, CtrlSkill reaches 64.5% Vrf accuracy, improving over the strongest baseline, Trace2Skill + Comb., by 26.5 percentage points, and achieves a 54.3% overall average. Without human-written initialization, CtrlSkill still obtains 55.0% Vrf accuracy and a 42.7% overall average in Creation, indicating that effective skills can be formed directly from execution feedback.

The same trend holds for DeepSeek-V4-Flash. CtrlSkill reaches 94.0%/89.0% Vrf/WikiTQ accuracy in Deepening and 89.5%/87.0% in Creation, with overall averages of 65.0% and 63.3%, respectively. These results show that the gains extend across initialization regimes, model families, and out-of-domain tasks, rather than being tied to a single evaluation metric.

To address the problems identified in Figure 1, we conduct ablations at three stages of skill self-evolution: update authorization, evidence timing, and execution grounding. These experiments examine how CtrlSkill controls skill updates through an outer feedback loop. The full method promotes three candidates and achieves the highest accuracy of 64.5%. This result indicates that CtrlSkill benefits not from increasing the number of active skills, but from selecting reliable policy updates.

Without admission control, candidates enter the active policy without passing the numerical evidence gate. The number of promoted candidates increases to eight, but accuracy drops to 45.0%, a decrease of 19.5 percentage points; one update is subsequently superseded. This shows that, without update authorization, locally plausible candidates with insufficient evidence can directly affect the active policy, and increasing the number of updates does not ensure better adaptation.

Without evidence accumulation, only two candidates are promoted and thirteen are rejected. Accuracy decreases to 47.5%, a drop of 17.0 percentage points. This setting weakens the temporal integration of feedback and causes the controller to make decisions from limited evidence, indicating that cross-batch accumulation controls the timing of policy updates.

Without tool-grounded constraints, four candidates are promoted and six are rejected, while accuracy falls to 46.0%, a decrease of 18.5 percentage points. Although candidates are still subject to admission decisions, the absence of constraints on tool execution, artifact generation, and result

read-back means that linguistically plausible experience may not become executable and verifiable behavior.

Overall, the three ablations disrupt different components of the feedback loop: update authorization, evidence timing, and execution grounding. CtrlSkill is therefore not merely a mechanism for increasing the number of skills, but a controllable skill-evolution system organized around observation, decision, update, and subsequent feedback. In Depth Analysis of Each Module in the Appendix A.4.

5 Further Analysis

Does Delayed Admission Select Reusable Experience? Training produced six candidates and 18 matched-task validations, but only C1, C2, and C4 were promoted. Their repair rates were 1/3, 1/4, and 5/5, respectively. C3 succeeded once but lacked sufficient support, C5 repaired only 1/5 tasks, and C6 received no subsequent match. These outcomes show that delayed admission separates candidate generation from policy activation: a local success becomes active only after later evidence supports its reuse. The complete candidate trajectories and the C1 state transition are reported in Appendix A.1.

Table 4: Skills authored by GPT 5.5 are tested with matched GPT 5.5 users; it report ANLS and Acc.

Method	ANLS	Acc. (%)
No Skill	0.8410	70.0
Trace2Skill + Comb.	0.8674	74.0
CtrlSkill	**0.9427**	**86.0**

How Does Admitted Experience Change Task Performance? On the same 200 held-out SpreadsheetBench instances, CtrlSkill retains 47 of the 66 Human-Written successes and repairs 82 of its 134 failures. The corresponding Trace2Skill+Comb. counts are 33 and 43. Thus, CtrlSkill achieves 71.2% success retention and 61.2% failure repair, compared with 50.0% and 32.1% for the baseline. A representative workbook case shows that the gain comes from making evaluator-visible values available in the saved artifact, rather than merely writing the correct formulas. The full instance-level comparison and skill requirements are given in Appendix A.2.

Can the Experience Be Reused by Another Model? We conducted an evolution of cross-model transfer capabilities on Kimi K3. As shown in Table 3, CtrlSkill consistently outperforms the No-Skill and Trace2Skill+Comb. baselines under both source-model settings. These results indicate that the learned experience transfers across model families and is not restricted to the response patterns of the source model.

Transfer to DocVQA. As shown in Table 4, CtrlSkill achieves an ANLS of 0.9427 and an accuracy of 86.0%, outperforming both No Skill (0.8410/70.0%) and Trace2Skill + Comb. (0.8674/74.0%). This corresponds to improvements of 0.1017 ANLS and 16.0 percentage points in accuracy over No Skill, and 0.0753 ANLS and 12.0 percentage points over Trace2Skill + Comb. The consistent gains across both metrics suggest that the evolved skills improve not only answer matching quality but also exact answer correctness, indicating transfer beyond the spreadsheet-oriented tasks used during evolution.

Comparison with existing skill-evolution methods. CtrlSkill is evaluated in the same SpreadsheetBench setting as Trace2Skill and SkillCAT, while differing in how execution evidence, candidate skills, and policy updates are handled. A concise comparison of their update mechanisms and policy-commitment strategies is provided in Appendix A.3, Table 7.

6 Conclusion

We presented CtrlSkill, a control-inspired framework for controllable skill self-evolution. It separates evidence observation, candidate editing, and policy admission, so that execution-derived updates affect the active policy only after validation on subsequent matching tasks. Experiments on SpreadsheetBench show consistent gains across models and initialization settings, while ablations support the roles of evidence accumulation, execution grounding, and delayed admission. Cross-model reuse with Kimi K3 further suggests that admitted skills can transfer when expressed as explicit operational constraints.

7 AI USE STATEMENT

Generative AI tools were used to assist with academic writing and language editing, the search for and identification of related work, and parts of the research ideation and execution. The tools served only as assistants and did not replace the authors' research judgment, experimental decisions, or final interpretation. All AI-assisted content and research results were manually checked, verified, and revised by the authors.

REFERENCES

Karl Johan Åström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. 2008.

Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Yang You, and Ivor W Tsang. Skilldag: Self-evolving typed skill graphs for llm skill selection at scale, 2026. *URL https://arxiv. org/abs/2606.03056*.

Nikolas Belle, Dakota Barnes, Alfonso Amayuelas, Ivan Bercovich, Xin Eric Wang, and William Wang. Agents of change: Self-evolving llm agents for strategic planning. *arXiv preprint arXiv:2506.04651*, 2025.

Kunfeng Chen, Qihuang Zhong, Juhua Liu, and Bo Du. Skillcat: Contrastive assessment and topology-aware skill self-evolution for llm agents. *arXiv preprint arXiv:2606.13317*, 2026.

Hongjun Liu, Yifei Ming, Shafiq Joty, and Chen Zhao. Harnessing llm agents with skill programs. *arXiv preprint arXiv:2605.17734*, 2026a.

Hongyi Liu, Haoyan Yang, Tao Jiang, Bo Tang, Feiyu Xiong, Yuyu Luo, and Zhiyu Li. Skillsvote: Lifecycle governance of agent skills from collection, recommendation to evolution. *arXiv preprint arXiv:2605.18401*, 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.

Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 2199–2208. IEEE, 2021.

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.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1470–1480, 2015.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

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.

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, 2023. *URL https://arxiv. org/abs/2305.16291*, 2(11), 2023.

Zefeng Wang, Minxi Yan, Jinhe Bi, Sikuan Yan, Volker Tresp, and Yunpu Ma. Metaskill-evolve: Recursive self-improvement of llm agents via two-timescale meta-skill evolution. *arXiv preprint arXiv:2607.05297*, 2026.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

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.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. 2026:104223–104294, 2026b.

Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Library drift: Diagnosing and fixing a silent failure mode in self-evolving llm skill libraries. *arXiv preprint arXiv:2605.19576*, 2026c.

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.

Table 5: Training record of C1, from failure evidence to skill activation.

Stage	Instance-level evidence or operation	Outcome and state
Batch 2: source failures	`41_47`: A1529 lacks "TOTAL"; `263_1`: H2 is empty instead of 3710; `267_18`: G1 is empty instead of the transformed string.	Three weak trajectories, each with score 0.
Candidate generation	Distinguish executing an operation from materializing its result in the final file. Require post-save value read-back and valid caches for formula outputs.	C1 created but not activated.
Batch 3: matching	Compare the original policy with the candidate-augmented policy on a matching task.	One task accumulated; pending.
Batch 4: accumulation	Continue paired evaluation on a new matching task.	Two tasks accumulated; pending.
Batch 5: validation	Three tasks accumulated; `84_40` improves from 0 to 1, while the other two remain unrepaired.	Repair rate 1/3; mean gain 0.3333.
Admission	Meet the configured gates, including at least three matching tasks, at least two original failures, repair rate ≥ 0.25, and mean gain ≥ 0.02.	C1 admitted into the active policy.

Table 6: Relevant skill requirements and observed execution for instance 52917.

Method	Relevant requirements	Observed execution and output
Human-Written	Use formulas, run recalculation after saving, and check formula errors.	Writes `DATE` formulas, but J2:J9 lack cached values.
Trace2Skill+Comb	Retains recalculation and verification requirements, including formula-mode and value-mode reads.	Writes the same formulas; final caches remain empty.
Ours	Check target locations, actual values, and semantic correctness in the final file.	Preserves formulas, populates date caches, and verifies formulas and values at the final path.

A Appendix

A.1 Delayed Admission and Candidate Reuse

A local failure can suggest a repair, but does not establish that the repair generalizes to other instances. We therefore separate candidate generation from skill activation and base admission on paired evaluations on subsequent matching tasks. Training produced six candidates and 18 matched-task evaluations, with three candidates promoted. As shown in Figure 4(b), C1, C2, and C4 repaired 1/3, 1/4, and 5/5 matching failures, respectively. C3 succeeded on its only matching task but lacked sufficient supporting instances; C5 repaired 1/5 tasks and did not pass admission; C6 received no matching instances. Admission thus depends on both the amount of evidence and the observed benefit, rather than the occurrence of a single success.

Table 5 traces the state transitions of C1. Its source tasks involve a summary row, a numerical calculation, and a string transformation. Despite these different operations, all three exhibit the same symptom: a required target cell lacks a readable value in the delivered workbook. The resulting candidate requires reopening the saved file and checking both the target location and its actual value. After batches 3 and 4, the accumulated number of matching tasks was one and two, respectively, and the candidate remained pending. It was admitted only in batch 5, after three matching tasks had accumulated and one failure had been repaired. A single promotion decision therefore aggregates evidence across multiple tasks; it does not activate experience directly from a single failure.

(a)

A: Day	B: Month	C: Year
1	1	2021
2	1	2021
3	1	2021
4	1	2021
5	1	2021
6	1	2021
7	1	2021
8	1	2021

J2:J9 = DATE(Cr, Br, Ar)

(b)

Cell	Human-Written	Trace2Skill+Comb	Ours
J2	None	None	2021-01-01
J3	None	None	2021-01-02
J4	None	None	2021-01-03
J5	None	None	2021-01-04
J6	None	None	2021-01-05
J7	None	None	2021-01-06
J8	None	None	2021-01-07
J9	None	None	2021-01-08
Official evaluation:	Fail	Fail	Pass

Figure 3: Cell-level comparison of delivered workbooks for instance 52917. Empty cached values occur in the two baseline outputs, whereas CtrlSkill provides evaluator-visible date values.

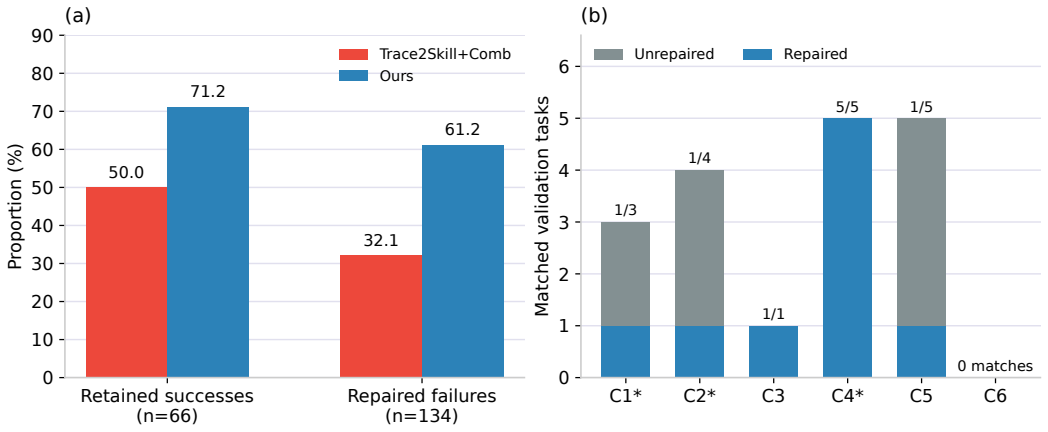

Figure 4: Test performance and training-time candidate validation. (a) Success retention and failure repair relative to Human-Written. (b) Matching validation tasks per candidate; blue denotes repaired tasks and gray denotes unrepaired tasks.

A.2 Performance Changes and Artifact Verification

We compare the success sets of the three methods on the same 200 test instances and then examine a representative task in detail. Taking the 66 tasks solved by Human-Written and its 134 failures as reference groups, Trace2Skill+Comb. retains 33 successes and repairs 43 failures, whereas CtrlSkill retains 47 successes and repairs 82 failures. Thus, CtrlSkill achieves a success-retention rate of 71.2% and a failure-repair rate of 61.2%, compared with 50.0% and 32.1% for Trace2Skill+Comb.

Figure 4(a) summarizes these instance-level transitions, while Figure 3 provides a concrete example of the execution-level difference. Instance 52917 requires combining the day, month, and year in columns A, B, and C into dates in cells J2:J9. All three methods write the same `DATE` formulas, but the two baseline workbooks contain empty cached values after saving. CtrlSkill, in contrast, preserves the formulas, materializes the corresponding date values, and verifies the saved workbook in both formula and value modes. The official evaluation therefore reports fail, fail, and pass, respectively.

Table 6 explains this difference by aligning the stated skill requirements with the observed execution and output. Human-Written already requires formula construction, recalculation, and formula-error checking, but its output still lacks evaluator-visible cached values. Trace2Skill+Comb. retains these requirements and additionally distinguishes formula-mode from value-mode reads; nevertheless, the final caches remain empty in this case. CtrlSkill explicitly requires checking target locations, actual

Table 7: Comparison of representative skill self-evolution methods.

Method	Experience source	Candidate assessment	Execution grounding	Policy commitment
Trace2Skill	Execution trajectories	Trajectory-level lesson distillation and consolidation	Primarily trajectory and outcome signals	Skill updates are consolidated after candidate generation
SkillCAT	Execution trajectories and candidate replay	Contrastive assessment and replay-based comparison	Outcome-level feedback with candidate evaluation	Candidate selection through comparative assessment
CtrlSkill	Structured execution evidence	Paired active/trial evaluation on subsequent matching tasks	Separates reasoning, tool behavior, artifact state, and acceptance	Evidence-gated accept, pending, or reject decisions

values, and semantic correctness in the final artifact, and its execution satisfies these requirements by populating and reading back the date values.

Together, the figure and table show that the relevant distinction is not whether a method can construct the correct formula, but whether the required result is materialized and observable in the delivered artifact. The case therefore provides behavioral evidence for the final policy: experience is expressed as an executable delivery constraint that connects tool execution, artifact state, and evaluator-visible output.

A.3 Comparison with Existing Skill-Evolution Methods

Table 7 positions CtrlSkill relative to representative skill self-evolution methods. Although Trace2Skill, SkillCAT, and CtrlSkill are evaluated on SpreadsheetBench, they make different design choices about how execution experience is converted into persistent skills.

The main distinction of CtrlSkill is it combines three constraints in a single update loop. First, execution observations are organized into evidence states that distinguish intended operations, actual tool execution, artifact state, and final acceptance. Second, each candidate is represented as a scoped patch with supporting evidence, applicability conditions, tool requirements, verification criteria, and exclusions. Third, the candidate is evaluated against the active policy on subsequent matching tasks before it can modify the active policy. The resulting decision depends on accumulated evidence, baseline failures, repair coverage, regression risk, and mean gain.

This design gives the methods different operational emphases. Trace2Skill primarily addresses how trajectory-local lessons can be distilled into transferable skills. SkillCAT emphasizes comparative assessment and candidate replay. CtrlSkill focuses on controlling the evidence, scope, and timing of persistent policy updates. The methods are therefore complementary: the trajectory distillation and replay mechanisms used by prior work can serve as candidate-generation or candidate-assessment components within the broader evidence-constrained admission loop of CtrlSkill.

A.4 Deep Ablation Analysis

Without admission control

Among the 63 instances where Full succeeds but this ablation fails, 35 failures are caused by missing output files and 28 by value mismatches. This failure pattern shows that removing numerical admission allows insufficiently supported candidate experiences to enter the active policy.
Representative case: In case `1818`, the task requires identifying the student with the lowest score and writing the result into `Summary!B3:C68`. The expected value in `B3` is `MR21_007`, whereas the ablated run leaves the cell empty. The Full system rejects the candidate unless its numerical evidence passes the admission gate, preventing an unsupported update from being propagated. Without this gate, the candidate is promoted despite incomplete execution, which leads to an incomplete final workbook.

Without evidence accumulation

Among the 53 instances where Full succeeds but this ablation fails, 21 failures are caused by missing output files and 32 by value mismatches. The larger number of value mismatches indicates that isolated evidence is insufficient for verifying multi-step spreadsheet transformations.
Representative case: In case `54196`, the task requires filling `Sheet two!E2:E279` using a lookup operation. The expected value in `E2` is `0D`, but the ablated run produces `#N/A`. Without evidence accumulation, the system accepts a locally plausible lookup procedure without checking whether the source range, matching key, and propagated results are mutually consistent. The Full system accumulates these signals and rejects the update when the evidence chain is inconsistent.

Without tool-grounded constraints

Among the 62 instances where Full succeeds but this ablation fails, 24 failures are caused by missing output files and 38 by value mismatches. The predominance of value mismatches shows that unconstrained tool use produces executable but semantically incorrect spreadsheet operations.
Representative case: In case `3911`, the task requires allocating 120 currency notes according to salary values. The expected value in `H4` is `2`, but the ablated run writes `3`. The operation completes without a file-level error, yet the allocation violates the task constraint. Tool-grounded constraints in the Full system restrict the editable range and enforce the intended numerical relationship, preventing this type of semantically valid but incorrect update.

A.5 Structured Analysis

For each failed instance, the system constructs a structured evidence record that connects the task requirement, workbook state, tool behavior, target region, final artifact, and official evaluation result. The first three panels describe the formal evidence-chain types retained by the controller. The fourth panel presents a composite case in which multiple evidence-chain types are activated simultaneously.

evaluator_visible_spreadsheet_value_materialization

Definition. This evidence-chain type concerns cases in which an artifact or an editing operation exists, but the evaluator cannot observe the required value in the final workbook.
Representative case: `case_263_1`. The task required the value `3710` in `Sheet1!H2`. The workbook was created and remained readable, and formulas were detected in the artifact. However, the official evaluator observed `None` at the target cell.
Failure cause. The agent treated the presence of a formula and the execution of a spreadsheet operation as sufficient evidence of completion. It did not establish that the formula had produced a non-empty evaluator-visible cached value in the saved workbook.
Extracted evidence. The evidence record explicitly contained:

- official evaluator message: `Value mismatch at H2: expected '3710', got 'None'`;
- `artifact_exists=true`;
- `artifact_readable=true`;
- formulas detected in three cells;
- `derived_operation_present=true`;
- `derived_value_materialized=false`;
- `derived_value_missing=true`;
- `formulas_present_but_cached_values_missing`.

These observations were linked as

formula or operation present → derived value not materialized → target value missing after save.

Resulting experience. The promoted experience requires the agent to construct a workbook evidence card before editing, link source ranges and rules to destination cells, and treat target-cell materialization as a mandatory checkpoint. Every requested target cell must contain either a literal value or a formula with a verified non-empty cached value. After saving, the exact final file must be reopened in value mode. If a required target remains empty, the agent must either use a supported recalculation path and verify its effect, or write a verified literal value before submission. Artifact existence, formula presence, and tool execution alone are not sufficient for delivery.

spreadsheet_rule_to_artifact_grounding

Definition. This evidence-chain type concerns cases in which the workbook is generated and the target cells are populated, but the natural-language task rules are not correctly grounded in the source workbook and destination locations.

Representative case: `case_22_47`. The task required data from columns A:C to be reordered into the output region F:H according to a helper list in column J. Matching rows had to preserve the helper-list order, while non-matching rows had to follow the specified fallback ordering. Blank rows, headers, and duplicates also had to be handled explicitly. The official evaluator reported:

```
F2:  expected '1', got '5'.
```

Failure cause. The workbook was readable and the target cell was non-empty, so the failure was not caused by a missing artifact or an empty formula cache. The code record states that the current spreadsheet state, final formula/output, and conflicting output and sorting instructions were not resolved before writing.

Extracted evidence. The evidence card recorded the following concrete issues:

- the current spreadsheet data in columns A:C was not fully grounded;
- the helper list in column J was not grounded;
- the intended output range was unresolved between F:H and G:H;
- the priority between helper-list ordering and final sorting was unresolved;
- the final output and formula state were not fully observed;
- the official evaluator observed `F2=5` instead of `F2=1`;
- `chain_complete=false`;
- the candidate action was `revise`, rather than `accept`;
- the screen reason explicitly reported missing actual spreadsheet state and unresolved output/sort instructions.

The resulting evidence relation was

$$\begin{aligned}\text{task rules} &\rightarrow \text{source and helper ranges} \rightarrow \text{precedence and ordering} \\ &\rightarrow \text{target range} \rightarrow \text{incorrect non-empty value.}\end{aligned}$$

Resulting experience. The promoted experience requires the agent to inspect columns A:C and helper column J, resolve the exact destination range, formalize helper priority and fallback sorting, and specify blank, header, and duplicate handling. Before editing, the agent must construct an intermediate output table or explicit source-to-target map. The first target cell and the expected first output rows must be known before modification. After saving, the agent must read back the exact target range and verify values, row count, ordering, headers, blanks, duplicates, and preserved regions. If the required workbook or rule evidence is unavailable, the agent must state the limitation rather than guess.

evaluator_visible_derived_value_materialization_gap

Definition. This evidence-chain type concerns cases in which a derived computation is performed or represented in the artifact, but its evaluator-visible result is not completed, saved, or verified in the final delivered file.
Representative case: `case_12307`. The official evaluator expected the value `2` in `I12`, but observed `None`. The workbook existed and opened successfully. Cells `I12:I13` contained formulas, but none of the formula cells had a valid cached value.
Failure cause. The execution produced a derived operation, but the final artifact did not contain the materialized result required by the evaluator. The presence of formulas and the existence of the workbook were incorrectly treated as proof that the derived computation had been delivered.
Extracted evidence. The code record marked the case with:

- `artifact_exists=true`;
- `artifact_readable=true`;
- formulas present in `I12:I13`;
- zero formula cells with valid cached values;
- two formula cells with blank cached values;
- `derived_operation_present=true`;
- `derived_value_missing=true`;
- `possible_materialization_gap=true`;
- `verification_trace_observed=true`;
- `official_failure_observed=true`;
- official evaluator message: `Value mismatch at I12:  expected 2, got None`.

The evidence chain was organized as

derived operation → recalculation or materialization → final save
→ reopen exact output path → value read-back → verification.

Resulting experience. The promoted experience adds a final-artifact materialization checkpoint after formulas, macros, transformations, or computations are written. The agent must distinguish formula presence, materialized evaluator-visible value, and blank or erroneous materialized value. If recalculation is used, the exact delivered file must be reopened and `I12:I13` must be read in value mode. Recalculation logs, formula text, tool messages, and intermediate files do not count as verification unless the final delivered artifact contains the required values.

Composite Evidence Case: `case_52917`

Case description. The task required combining day, month, and year values from columns A, B, and C into date values in column J. In the final run, the official evaluator reported success, but the evidence card still showed a composite routing case.

Failure evidence inherited from the execution history. The record attached the following promoted patches:

- `patch_b002_evidence_cluster_002`;
- `patch_b005_evidence_cluster_001`;
- `patch_b013_evidence_cluster_004`.

The classification record stored:

- `primary_evidence_chain_type = candidate_new_chain_type`;
- secondary types: `evidence_cluster_002`, `evidence_cluster_001`, and `evidence_cluster_004`;
- `candidate_new_chain_type = unclustered_evidence_gap`;
- `classification_confidence=0.5`;
- `evidence_quality=0.5`;
- `chain_complete=false`;
- missing evidence nodes: `observed_gap`, `why_existing_types_fail`, and `candidate_required_nodes`;
- repair suggestions requiring explicit evidence locators and an evidence edge connecting observations to the conclusion.

Evidence organization. The case was not copied into three independent experiences. Instead, the controller retained one primary route and multiple secondary mechanisms:

$$\begin{aligned} \texttt{primary_evidence_chain_type} &\leftarrow \text{main routing decision},\\ \texttt{secondary_evidence_chain_types} &\leftarrow \text{additional interacting mechanisms},\\ \texttt{evidence_units} &\leftarrow \text{grounded task and artifact observations},\\ \texttt{evidence_edges} &\leftarrow \text{relations linking evidence to conclusions}. \end{aligned}$$

Resulting composite experience. The final experience combines rule grounding, evaluator-visible value materialization, and derived-value verification. It requires the agent to identify the source columns and target range, construct the date formulas, materialize the resulting dates, save the exact final artifact, reopen it, and verify both formulas and values. Candidate validation is then performed on the combined experience rather than by separately promoting three duplicate skills.