REPAIRGATE: RL-Trained Skill Repair and Abstention for Lifelong Agent Libraries

Anonymous ACL submission

Abstract

LLM agents increasingly rely on externalized skill libraries, yet existing maintainers typically delegate skill editing to a prompted LLM, leaving the question of how to *train* a skill maintainer open. We present REPAIRGATE, a 9B drop-in wrapper around the standard candidate patcher: a counterfactually-trained post-trial gate accepts, blocks, or overrides each candidate patch, with a structured repair protocol for field-level edits and overlap-aware regularization to control library growth. Across six base agents on SKILLFLOW (Zhang et al., 2026b), REPAIRGATE delivers +7.73 pp over Vanilla and +9.13 pp over prompted +Iter, with the largest gains on the weakest base agents where prompted iteration regresses, at a standardized $0.033/trial local overhead on top of the shared candidate-patching pipeline. Code and data: https://anonymous.4open.science/r/RepairGate-F564.

1 Introduction

Autonomous agents increasingly accumulate *skills*—reusable procedural artifacts that persist across tasks (Wang et al., 2023; Zheng et al., 2025; Anthropic, 2026). The pattern is standard in deployed coding and data agents (Anthropic, 2025; OpenAI, 2025): at inference the agent retrieves relevant skills, and after each task a separate LLM proposes edits to the shared library from the trajectory and verifier feedback.

The maintenance gap. SKILLFLOW reports three structural failure modes—*drift*, *inflation*, and a *repair gap* (fixing bad skills, not writing them, is the dominant bottleneck)—across 11 agent-model stacks on 166 workplace tasks. Across this landscape the editor is still typically a prompted teacher; even SkillRL, the only prior RL approach in this space, trains a 7B policy to *use* skills while leaving the editor prompted (Figure 1).

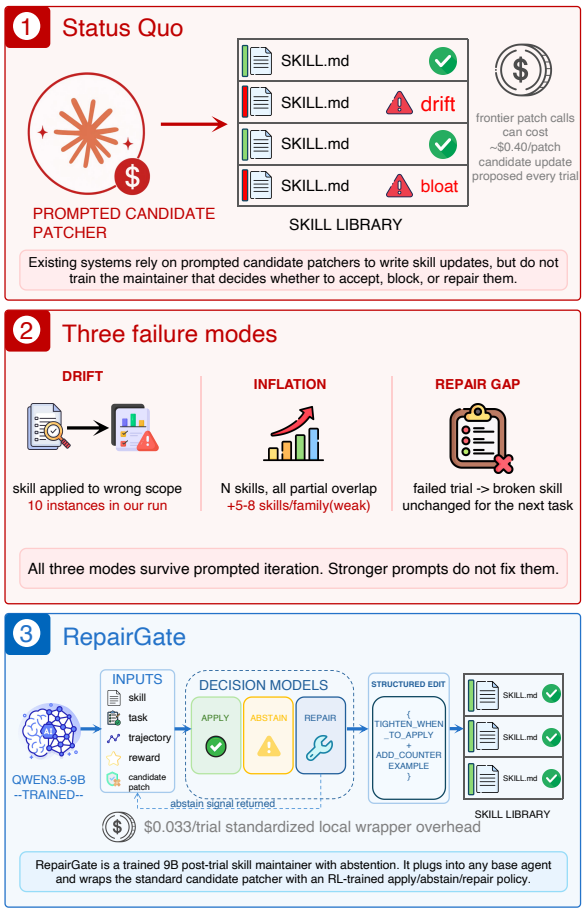

Figure 1: The skill-library maintenance gap. **Top:** prompted candidate patchers propose per-trial skill updates, with frontier calls costing roughly $0.40/patch. **Middle:** three failure modes from SKILLFLOW: drift, inflation, and repair gap. **Bottom:** REPAIRGATE wraps the candidate patcher with a trained 9B post-trial APPLY/ABSTAIN/REPAIR maintainer, adding $0.033/trial local overhead.

Our Opus 4.5 run surfaces 10 live binary-drift instances ($R_{\text{with}} < R_{\text{without}}$); a fine-grained Medical-Data-Standardization rerun (Appendix C) reveals the same mechanism below the binary level (per-assertion pass 61% → 11%) even when aggregate iter is overall positive (+3.01 pp). Aggregate it-

eration thus leaves systematic residuals a trained maintainer could remove.

REPAIRGATE. We present REPAIRGATE, a 9B Qwen3.5-LoRA (Qwen Team, 2026) policy served via vLLM (Kwon et al., 2023) and trained with GRPO to jointly **repair** skills through a structured repair protocol over `SKILL.md`, **abstain** via a counterfactually-trained gate over $\{\text{APPLY}, \text{ABSTAIN}, \text{REPAIR}\}$ with asymmetric penalties, and **control growth** via overlap-aware reward regularization. The gate, repair protocol, and overlap-aware regularizer respectively target drift, the repair gap, and inflation.

Contributions.

1. **Insight & Formulation.** We formally decouple skill drift detection from task resolution in agentic lifelong learning. The detection meta-task admits a counterfactual sign-classification formulation under $\Delta_t^{\text{cf}} = R^{\text{with}} - R^{\text{without}}$, making skill repair tractable for a small trainable policy without solving the underlying tasks.

2. **Methodology.** We propose REPAIRGATE, the first RL-trained post-trial skill maintainer that learns to accept, block, or override candidate skill patches. It combines a counterfactually-trained abstention gate with asymmetric drift-cost penalties and a structured repair protocol over `SKILL.md` that enables per-action attribution and ablation.

3. **Performance & Efficiency.** Across six base agents on SKILLFLOW, the same 9B drop-in wrapper delivers +7.73 pp over Vanilla and +9.13 pp over prompted +Iter, outperforms five skill-management baselines on Opus 4.6, and adds a standardized \$0.033/trial local maintainer overhead on top of the shared candidate-patching pipeline.

2 Related Work

Skill libraries and lifelong skills. LLM agents maintaining an explicit reusable skill library trace back to Voyager and SSO (Nottingham et al., 2024), with industrial instantiations in Claude skills and Codex CLI. SkillWeaver, SKILLFLOW, SkillCLAW (Ma et al., 2026), MemSkill (Zhang et al., 2026a), and Trace2Skill (Ni et al., 2026) extend the loop with trajectory-local distillation, episodic/procedural memory (Zhao et al., 2024; Sumers et al., 2023; Wu and Zhang, 2026), and verifier-grounded patching, with software-engineering instantiations (Yang et al., 2024). Across this landscape, the component that decides how to *edit* a skill is typically a prompted large teacher.

Agentic RL. Building on agentic-LLM bootstrapping (Yao et al., 2022; Wei et al., 2022) and RLHF/process supervision (Ouyang et al., 2022; Lightman et al., 2024), recent agentic-RL methods (Qi et al., 2025; Wei et al., 2025; Song et al., 2024; Li et al., 2025; Bai et al., 2024) and RL infrastructures (Shao et al., 2024; Sheng et al., 2025; Hu et al., 2024; Guo et al., 2025) train task-execution policies. In the skill-based setting, SkillRL (Xia et al., 2026) is the only prior RL approach; it trains a 7B policy to *use* skills produced by a prompted *o*3 teacher, leaving the editing decision unsupervised. We instead train the post-trial maintainer that accepts, blocks, or overrides candidate skill patches. Tool-use and GUI-agent work (Schick et al., 2023; Qin et al., 2024; Gou et al., 2025; Patil et al., 2024) typically treats tool catalogs as static; our focus is the evolving procedural skill library.

Self-correction, verification, abstention. Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) iteratively rewrite outputs but do not gate library maintenance. CRITIC (Gou et al., 2024) and Huang et al. (2024) show intrinsic self-correction without external feedback is unreliable; principled deferral (Kossen et al., 2022; Hosseini et al., 2024), LLM-as-judge (Zheng et al., 2023), and prompted disagreement gates (Tang et al., 2025) ground deferral externally. Our contribution is the analogous mechanism for skill-library *maintenance*, RL-trained with counterfactual rollouts and asymmetric penalties reflecting the cost structure of drift.

REPAIRGATE is distinguished by four jointly-held properties: a trained post-trial maintainer for candidate skill patches; a gate specific to skill-library maintenance; counterfactual supervision from explicit with-/without-skill rollouts; and structured repair actions with per-action attribution rather than free-form rewriting.

3 Method: REPAIRGATE

3.1 Problem Setup

We adopt the Agentic Lifelong Learning protocol of SKILLFLOW. A task family $\mathcal{F} = (T_1, \ldots, T_n)$

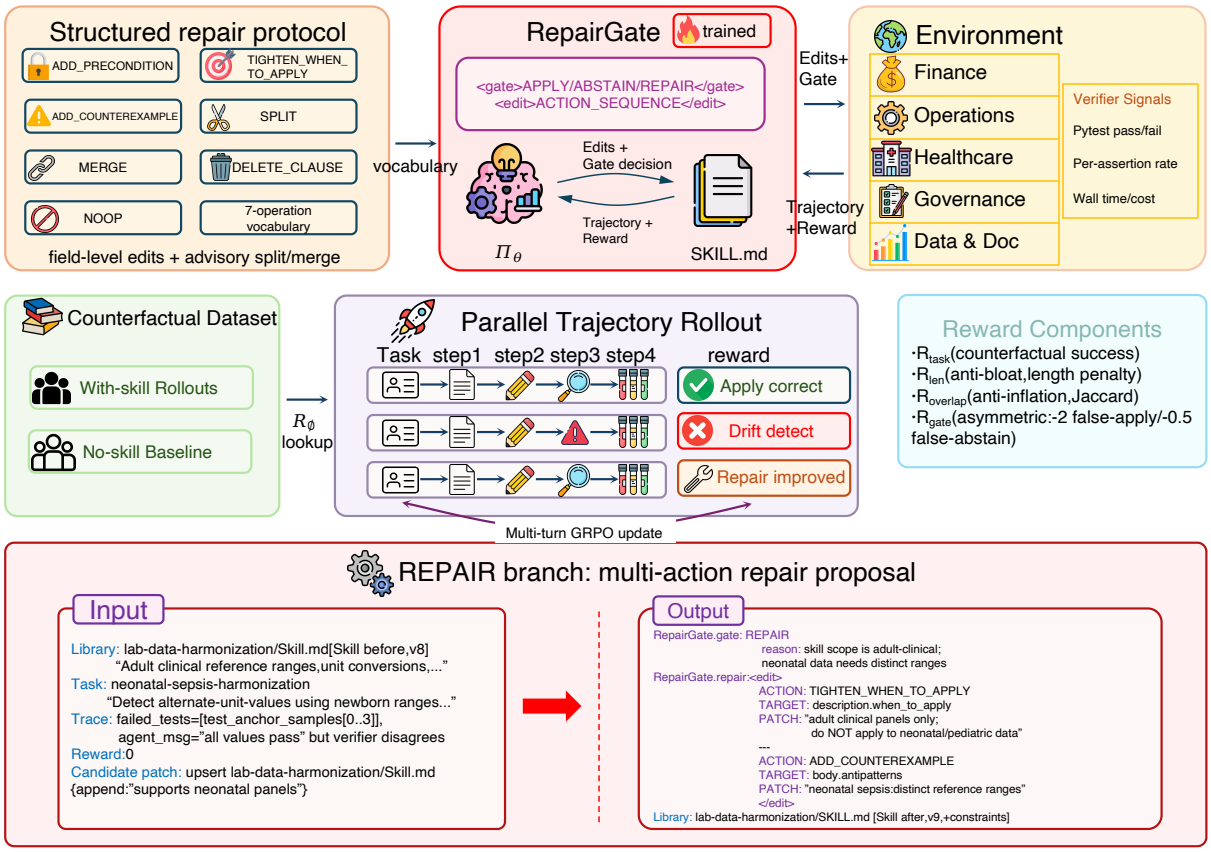

Figure 2: REPAIRGATE overview. **Top:** REPAIRGATE wraps the standard candidate patcher, predicts APPLY/ABSTAIN/REPAIR, and emits structured repair proposals only on REPAIR trials. **Middle:** GRPO evaluates sampled updates by parallel rollouts, with precomputed $R_t^{\emptyset}$ for counterfactual normalization. **Bottom:** worked drift case: given the library, task, trace, reward, and Δ_t^{cand}, REPAIRGATE predicts REPAIR, tightens `when_to_apply`, and adds an anti-pattern.

is presented in fixed order; the skill library before T_t is $\mathcal{S}_{t-1}$, the (frozen) base agent is A_ϕ, and the pytest verifier returns binary $R_t \in \{0, 1\}$. For training and analysis we collect paired with-library and no-library rollouts—$R_t^{\mathcal{S}} = V_t(A_\phi(T_t, \mathcal{S}_{t-1}))$ and $R_t^{\varnothing} = V_t(A_\phi(T_t, \varnothing))$—and define the *counterfactual skill gain*

$$\Delta_t^{\text{cf}} \;=\; R_t^{\mathcal{S}} - R_t^{\varnothing} \;\in\; \{-1, 0, +1\}, \qquad (1)$$

which serves as a rollout-level estimate of skill-induced transfer (negative ⇒ skill hurts; positive ⇒ skill helps). Let Δ_t^{cand} denote the candidate patch produced by SKILLFLOW's standard per-trial patcher (in our deployment, the base agent itself). We wrap this candidate patcher with a learned gate-and-repair policy

$$\pi_\theta(g_t, e_t \mid x_t), \quad x_t = (\mathcal{S}_{t-1}, T_t, \tau_t^{\mathcal{S}}, R_t^{\mathcal{S}}, \Delta_t^{\text{cand}}),$$

where $g_t \in \{\text{APPLY}, \text{ABSTAIN}, \text{REPAIR}\}$ and e_t is a sequence under the structured repair protocol (§3.3), and $\tau_t^{\mathcal{S}}$ is the agent's with-library trajectory on T_t. The library update routes between the candidate patch, identity, and a structured edit:

$$\mathcal{S}_t \;=\; \begin{cases} \text{Apply}(\Delta_t^{\text{cand}}, \mathcal{S}_{t-1}), & g_t = \text{APPLY}, \\ \mathcal{S}_{t-1}, & g_t = \text{ABSTAIN}, \\ \mathcal{U}(\mathcal{S}_{t-1}, e_t), & g_t = \text{REPAIR}. \end{cases} \qquad (2)$$

Thus APPLY forwards the candidate patch (which may upsert a new skill, modify an existing one, or delete a clause), ABSTAIN blocks it, and REPAIR replaces it with a structured edit sequence on the current library. **Skill creation enters through the APPLY branch**: if Δ_t^{cand} is an upsert for a new skill, accepting it creates the corresponding skill card. The repair head is intentionally restricted to structured modifications of the current library rather than open-ended skill discovery. The policy is a Qwen3.5-9B-Instruct backbone with LoRA adapters (Hu et al., 2022) and two heads: a **Gate head** π_θ^{gate} produces g_t, and a **Repair head** $\pi_\theta^{\text{repair}}$ is invoked only when $g_t = \text{REPAIR}$ (Figure 2). The repair head is invoked on only ∼30% of trials, keeping per-trial cost low (§4.2).

Post-trial gate. The gate is post-trial: the score of T_t is the first with-library rollout (computed before π_θ is queried), and the gate output only determines $\mathcal{S}_t$ for future tasks $T_{t+1:}$. Thus ABSTAIN is a maintenance decision, not a runtime safety switch. $R_t^{\varnothing}$ enters x_t only via proxy-target and reward construction during training, never at inference time.

3.2 Drift as Counterfactual Negative Transfer

We ground the architecture in the empirical structure of drift, drawn from 10 live binary-drift instances in our 166-trial Opus 4.5 teacher run that span 6 families and 6 distinct skill types (Figure 5; Appendix D expands two canonical binary-drift mechanisms (D2 calc-engine, D3 logic-specialization) and one additional fine-grained regression pattern (D1 format-substring) that does not register at the binary reward level).

For each skill s, write $\mathrm{App}(s)$ for the tasks on which the agent applies s and $\mathrm{Val}(s)$ for the unknown set on which s is actually correct. Drift arises when $\mathrm{App}(s) \supsetneq \mathrm{Val}(s)$: a later T_t inherits the skill outside its scope and the agent's computation is mathematically correct but skill-induced failure. Since $\mathrm{Val}(s)$ is unobserved, we treat $\Delta_t^{\mathrm{cf}} < 0$ as rollout-level evidence (not a certificate) that the library, not the agent, is the bottleneck. Drift detection thus becomes a sign-classification problem decoupled from solving T_t, making maintenance tractable for a 9B policy.

3.3 Structured Edit Action Space

Each skill card contains structured fields (description, `when_to_apply`, preconditions, anti-patterns, body). The repair head emits a short sequence of typed edits over these fields—we call this typed output format the *structured repair protocol*. Five labels are executable field-level edits in our applier: ADD_PRECONDITION, TIGHTEN_WHEN_TO_APPLY, ADD_COUNTEREXAMPLE, DELETE_CLAUSE, and NOOP. SPLIT_SKILL and MERGE_SKILLS are retained only as advisory labels for attribution and do not create or merge skill cards in our experiments. Full action semantics and the field-level applier are detailed in Appendix A.

3.4 Counterfactual Reward for the Gate

The binary R_t does not reveal whether a *skill* helped or hurt: an agent may succeed despite a misleading skill or fail despite a good one. We use Δ_t^{cf} plus a binary skill-causal indicator $c_t \in \{0, 1\}$ (set to 1 when $\tau_t^{\mathcal{S}}$ shows direct skill-misuse evidence from defect injection or manual review). We call this signal a *proxy gate label*, not an oracle action label, and write it $\tilde{g}_t$:

$$\tilde{g}_t = \begin{cases} \text{APPLY}, & \Delta_t^{\mathrm{cf}} > 0, \\ \text{REPAIR}, & \Delta_t^{\mathrm{cf}} < 0, \text{ or} \\ & \Delta_t^{\mathrm{cf}} = 0 \wedge R_t^{\mathcal{S}} = 0 \wedge c_t = 1, \\ \text{ABSTAIN}, & \text{otherwise.} \end{cases} \tag{3}$$

$\Delta_t^{\mathrm{cf}} > 0$ provides rollout-level evidence that the *current library* helped execution; it does not prove that the next candidate patch Δ_t^{cand} is globally optimal. We therefore interpret APPLY operationally as "do not interrupt the base-agent evolution step," biased by the proxy signal that current evolution has been helpful. The negative-transfer ($\Delta_t^{\mathrm{cf}} < 0$) and symmetric-failure-with-skill-evidence cases map to REPAIR; everything else (irrelevant skills, task-side failures) maps to ABSTAIN. Length/overlap regularization and GRPO Docker re-scoring (§3.5) provide additional pressure against bloated or harmful writes that this proxy alone cannot rule out.

3.5 Reward Function

During GRPO training on the auxiliary corpus, after the policy emits (g_t, e_t) we apply (2) to obtain $\mathcal{S}_t'$ and re-score the task with the training base agent A_ϕ^{train} to get $R_t^{\mathcal{S}'} = V_t(A_\phi^{\mathrm{train}}(T_t, \mathcal{S}_t'))$; the no-library baseline $R_t^{\varnothing}$ used in R_{task} below is computed once per auxiliary prompt with the same A_ϕ^{train} so the counterfactual is apples-to-apples. (At evaluation the policy is run as a drop-in `SkillPatchEvolver` wrapper; no re-score is performed—see §4.1.) The total reward is

$$R = R_{\mathrm{task}} + \alpha R_{\mathrm{len}} + \beta R_{\mathrm{ovlp}} + \gamma R_{\mathrm{gate}}, \tag{4}$$

with $R_{\mathrm{task}} = R_t^{\mathcal{S}'} - R_t^{\varnothing}$ (counterfactual-normalized so drift on one task receives negative reward). This same-task re-score $R_t^{\mathcal{S}'}$ is used only as a tractable surrogate for library-update quality during training; evaluation remains first-attempt (§4.1), so post-trial edits are never allowed to change the score of the current evaluation task and can affect only subsequent tasks. For length and overlap regularization, let p_t denote the patch actually applied:

$$p_t = \begin{cases} \Delta_t^{\mathrm{cand}}, & g_t = \text{APPLY}, \\ \varnothing, & g_t = \text{ABSTAIN}, \\ e_t, & g_t = \text{REPAIR}. \end{cases}$$

	Vanilla		+Iter		+RepairGate (ours)		Gain	
Base Agent	%Comp↑	Cost↓	%Comp↑	Cost↓	%Comp↑	Cost↓	Δ^{V}	Δ^{I}
Claude Opus 4.6	53.61	0.741	54.82	0.689	**60.24**	0.732	+6.63	+5.42
Claude Sonnet 4.6	47.59	0.193	46.39	0.281	**55.42**	0.247	+7.83	+9.03
Claude Opus 4.5	46.39	0.385	49.40	0.437	**53.01**	0.416	+6.62	+3.61
Claude Sonnet 4.5	40.36	0.328	39.76	0.279	**48.19**	0.305	+7.83	+8.43
MiniMax M2.7	30.72	0.014	25.30	0.021	**39.76**	0.045	+9.04	+14.46
GLM 5.1	31.93	0.016	26.51	0.023	**40.36**	0.053	+8.43	+13.85
Average	41.77	0.279	40.36	0.288	**49.50**	0.300	**+7.73**	**+9.13**

Table 1: SkillFlow main results (166 tasks, 20 families). Cost is per-trial USD under the SkillFlow logging convention. +Iter and +RepairGate use the same candidate-patching pipeline; relative to +Iter, +RepairGate adds a standardized \$0.033/trial local A800 gate/repair overhead. The \$0.033 is an incremental local overhead, not a standalone end-to-end deployment cost. $\Delta^{\mathrm{V}} = \mathrm{RG} - \mathrm{V}$ and $\Delta^{\mathrm{I}} = \mathrm{RG} - \mathrm{I}$; RepairGate improves over Vanilla/+Iter by +7.73/+9.13 pp on average.

Then $R_{\mathrm{len}} = -\min(|p_t|_{\mathrm{tok}}/L_{\max}, 2)$. For the overlap penalty, let $\mathcal{A}_t$ be the set of skill cards created or modified by applying p_t and still present in $\mathcal{S}'_t$. For each affected card $s' \in \mathcal{A}_t$, define

$$h(s') = \max_{s_k \in \mathcal{S}'_t \setminus \{s'\}} J(s', s_k),$$

where J is token-level Jaccard and the maximum over an empty set is defined as 0. We set $R_{\mathrm{ovlp}} = 0$ for ABSTAIN or pure-deletion patches; otherwise, $R_{\mathrm{ovlp}} = -|\mathcal{A}_t|^{-1} \sum_{s' \in \mathcal{A}_t} h(s')$. Thus an affected card is never compared with itself. SPLIT_SKILL and MERGE_SKILLS are advisory labels and do not themselves create or merge cards; unless accompanied by executable field-level edits, they leave $\mathcal{A}_t = \varnothing$ and contribute nothing to R_{ovlp}. The gate reward $R_{\mathrm{gate}}(g_t, \tilde{g}_t) = M_{\tilde{g}_t, g_t}$ is read off the 3×3 penalty matrix

$$M = \begin{array}{c|ccc} \tilde{g}_t \backslash g_t & \text{APL} & \text{ABS} & \text{REP} \\ \hline \text{APL} & +1 & -0.5 & -0.5 \\ \text{ABS} & -2 & +1 & -0.3 \\ \text{REP} & -2 & -1 & +1 \end{array} \tag{5}$$

which encodes three asymmetries: predicting APPLY when the proxy label is REPAIR costs -2, the heaviest penalty, because the candidate patch is forwarded unchecked and a harmful update is committed to the library; predicting APPLY on an unsupported case (proxy label ABSTAIN) is also penalised -2 because it writes a patch that should have been blocked; false-abstain on positive transfer (proxy label APPLY) costs only -0.5 (the blocked patch is a missed write, recoverable on a later trial); under-repair (proxy label REPAIR, predicted ABSTAIN) costs -1 (the broken skill stays in place but no new harm is added). Default weights $(\alpha, \beta, \gamma) = (0.05, 0.10, 0.5)$; a $\times 0.5/\times 2$ sensitivity sweep is in Appendix I.

3.6 Training: SFT then GRPO

Training uses an *auxiliary* lifelong-learning corpus disjoint from the SkillFlow evaluation set: four independently authored workflow families (tabular cleaning, document transformation, stateful operations, synthetic finance) of eight tasks each. We run an Opus 4.5 lifelong-learning pilot on this corpus to collect counterfactual rollout pairs and teacher patches, augmented by controlled skill-defect injection over D1/D2/D3 and edge cases. **No artefact from the 166 SkillFlow evaluation trials appears in any training prompt** (residual mechanism-level overlap is discussed in Limitation **(L3)**). The cold-start SFT phase fine-tunes π_θ on $\sim$1,000 prompts ($\sim$500 gate + $\sim$500 repair) under standard token-level cross-entropy. The GRPO phase then optimises the policy on $\sim$500 prompts via veRL with group-relative advantage (group size $G = 8$) and a KL penalty against $\pi_\theta^{\mathrm{SFT}}$. During GRPO, environment reruns use a fixed training base agent $A_\phi^{\mathrm{train}} = \mathrm{GLM}\ 5.1$. This training agent is used only on the auxiliary corpus; although GLM 5.1 is also one of the six evaluation base agents in §4.1, no SkillFlow evaluation trajectory or reward is used during training. Each rollout applies the candidate edit via (2), A_ϕ^{train} re-runs T_t in Docker, and the verifier returns (4). Full data construction in Appendix E; hyper-parameters in Appendix F.

4 Experiments

4.1 Setup

We follow the SkillFlow Agentic Lifelong Learning protocol. Tasks within each family run in fixed difficulty order with the skill library resetting between families; all trials exe-

cute in Docker with a 30 min agent and 5 min verifier timeout, and binary reward is 1 iff the full pytest suite passes. Patch schema and prompt template follow SKILLFLOW verbatim ((summary, upsert_files, delete_paths) at temperature 0.2, max 16,384 tokens). All base agents run through Claude Code CLI v2.1.118; the trained REPAIRGATE policy is wrapped in a thin adapter exposing SKILLFLOW's `SkillPatchEvolver` interface and served by vLLM, invoked with byte-identical skill-loading semantics—from a base agent's perspective, REPAIRGATE is just a different `SkillPatchEvolver` wrapper: it obtains the standard candidate patch, then accepts, blocks, or overrides it. Integration is a single-line import change, enabling cross-agent transfer with no further training.

First-attempt evaluation protocol. Evaluation follows SKILLFLOW's first-attempt protocol verbatim: each T_t is scored on the first with-library rollout, and REPAIRGATE's library edits affect only subsequent tasks T_{t+i} in the same family. The rescore $R_t^{\mathcal{S}'}$ used to compute training reward is not used at evaluation time; no repair-triggered rerun of the current evaluation task is counted in Tables 1–4.

Base agents. We evaluate REPAIRGATE on six base agents grouped by parameter scale and generation: *frontier* (Opus 4.5/4.6), *mid* (Sonnet 4.5/4.6), and *weak* (MiniMax M2.7, GLM 5.1); this tier vocabulary is used throughout.

Candidate patcher. In Table 1, the candidate patcher follows the corresponding base-agent setting. Table 2 fixes Opus 4.6 as the executor for cross-method comparison; hence +Iter/+Voyager/+SSO/+AgentKB use Opus 4.6 patching, while +SkillRL follows its native 7B+Opus setup.

4.2 Main Results on SKILLFLOW

Table 1 reports task completion %Comp and per-trial USD cost across six base agents under three regimes: **Vanilla** (no skill evolution), **+Iter**, and **+REPAIRGATE**. REPAIRGATE's incremental local wrapper overhead is a standardized \$0.033/trial (\$0.008 gate call + 29.5%×\$0.085 repair-proposal call, §4.4).

Key takeaways.

1. *Prompted iteration is unreliable across capability tiers.* Frontier-tier agents gain +1.21 (Opus 4.6) to +3.01 pp (Opus 4.5); mid-tier hover within ±1.2 pp; weak-tier (MiniMax, GLM) regress by 5.4 pp each. The shared mechanism (§3.2) is the same on every base agent; what changes with capability is whether the patcher recognizes scope mismatch when writing the next patch, so drift compounds for weaker patchers and §4.5 reproduces the regression on benchmarks with no shared family structure.

2. REPAIRGATE *improves all six agents, with largest gains on the weakest.* The two weakest agents gain +9.04/+8.43 pp, exceeding +6.63 pp on Opus 4.6—the gate primarily cancels over-application that weak agents commit more often.

3. *Overhead is a small standardized local wrapper cost.* REPAIRGATE shares the candidate-patching pipeline with +Iter and only adds the local gate+repair wrapper: \$0.033/trial = \$0.008 gate (every trial) + 0.295×\$0.085 repair-proposal call (§4.4).

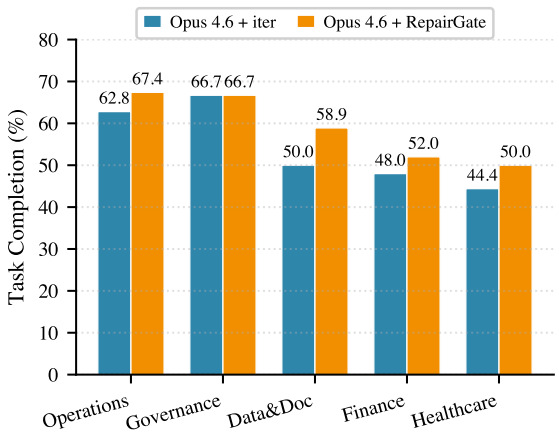

Figure 3: Per-domain completion on Opus 4.6. Trial-weighted means 91/166 (+Iter) and 100/166 (+REPAIRGATE) match Table 1. REPAIRGATE lifts the weak-domain tail by +4 to +9 pp.

Figure 3 breaks the gain down by domain on Opus 4.6. REPAIRGATE matches +Iter on the strongest two domains (Operations and Governance, near ceiling) and lifts the weak-domain tail by +5 to +6 pp on Data & Doc, Finance, and Healthcare.

Method	%Comp↑	Cost↓	Patcher
Vanilla	53.61	0.741	-
+Iter	54.82	0.689	Opus 4.6
+Voyager	54.22	0.762	Opus 4.6
+SSO	56.63	0.815	Opus 4.6
+AgentKB	54.22	0.823	Opus 4.6
+SkillRL	56.02	0.611	7B+Opus
+REPAIRGATE	**60.24**	**0.732**	**9B+Opus**

Table 2: Skill-management baselines on Opus 4.6. Cost is per-trial USD under the same SKILLFLOW logging convention as Table 1. REPAIRGATE outperforms the strongest prompted baseline, SSO, by +3.61 pp.

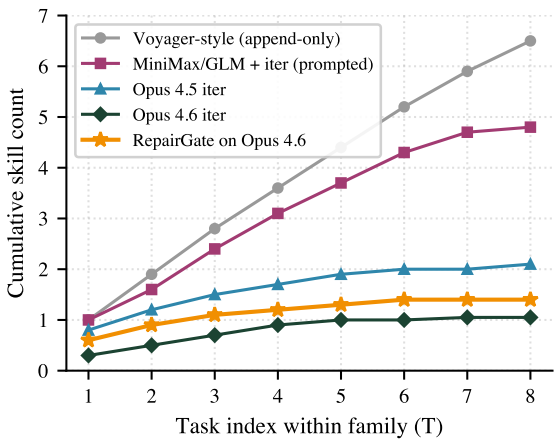

Figure 4: Cumulative skill count per task index (averaged over 20 families). Voyager-style append-only and weak-agent prompted +Iter grow monotonically to 5–8 skills/family; REPAIRGATE ends at 1–2 across all six base agents, matched only by prompted iter on Opus 4.6.

4.3 Comparison Against Skill-Management Baselines

Table 2 compares REPAIRGATE against five skill-management strategies on Opus 4.6: +Iter, +Voyager, +SSO, +AgentKB, and +SkillRL. Voyager/SSO use a prompted Opus teacher; AgentKB inserts a prompted disagreement gate without a repair head; SkillRL trains the skill *user* while leaving the writer prompted (we port their recipe to our SKILLFLOW data). REPAIRGATE shares the candidate-patch source with +Iter and adds only the local gate/repair wrapper ($0.033/trial standardized A800 accounting); +Voyager/+SSO/+AgentKB rely on Opus-class free-form patching for their prompted maintenance steps.

Why baselines fall short. Prompted editors (+SSO, +AgentKB) lack with-/without-skill counterfactual grounding, so false-apply and false-abstain mistakes are indistinguishable. SkillRL trains the skill *user* while leaving the editor prompted; our 60.24% vs. 56.02% comparison shows learning when to use skills does not close the repair gap—the policy must learn how to repair them.

4.4 Analysis

We probe REPAIRGATE along three axes on Opus 4.6: components, gate behavior, library growth.

Component ablation. Table 4 attributes the gain across components. Removing the gate drops %Comp by −9.64 pp: drift detection is the dominant contributor. "Gate-only on +Iter" disables the repair head and routes only between accept/block, climbing 54.82→56.02 (+1.20 pp over +Iter); the −4.22 pp drop vs the full system quantifies the value of replacing wrong candidate patches with structured edits. Outcome-only reward drops gate precision 0.81 → 0.62; removing length/overlap regularizers inflates #Skills to 2.9/3.4. The full 2.4% DriftRate decomposes into 1/166 missed baseline drifts (recall 0.90 over the 10/166 baseline) plus 3/166 new drifts REPAIRGATE's edits introduced. Two families (PPT-Formatting, Sales-Pivot) lie outside the structured repair protocol.

Gate behavior. On a held-out pool of the 10 canonical drift instances and the SFT validation split, asymmetric R_{gate} achieves 0.13 false-apply (vs 0.21 symmetric). Decision counts on the 166-trial benchmark are 82/35/49 APPLY/ABSTAIN/REPAIR (49.4/21.1/29.5%) with precisions 0.87/0.83/0.81; REPAIR concentrates where the binary verifier signal is strongest (62% on zero-pass families, 10% on stronger ones). Prompted AgentKB reaches only 0.62 precision on the same pool, suggesting counterfactual supervision is more reliable than self-rated confidence in this setting.

Repair head accuracy. On the 10 canonical drift cases (held out from SFT), action-set Jaccard is 0.71, exact-match 5/10, per-edit `TARGET` match 0.83. Errors concentrate on calc-engine cases where the gold annotation prefers the high-level SPLIT_SKILL diagnosis while v1 emits the executable field-level repair ADD_PRECONDITION.

Library growth. REPAIRGATE curbs library inflation across all six base agents; see Figure 4 for

	SkillsBench				SWE-Skills-Bench			
Base Agent	Vanilla	+Iter (static)	+Iter (evolve)	+REPAIRGATE	Vanilla	+Iter (static)	+Iter (evolve)	+REPAIRGATE
Opus 4.6	30.6	32.0	24.5	**37.4**	39.5	41.1	28.6	**46.2**
Sonnet 4.6	26.8	27.5	19.4	**33.1**	33.4	35.3	23.8	**40.8**
Opus 4.5	22.0	21.6	14.7	**28.4**	31.6	33.3	22.5	**40.0**
Sonnet 4.5	17.3	15.2	10.9	**23.7**	30.6	32.7	22.5	**38.7**
Average	24.2	24.1	17.4	**30.7**	33.8	35.6	24.4	**41.4**

Table 3: Cross-benchmark transfer with two iter conditions on the four Claude agents from Table 1 (MiniMax/GLM omitted for cost). Static +Iter $+1.8$ pp on SWE-Skills-Bench, flat on SkillsBench; LLM-evolved +Iter *regresses* $\sim-7/-9$ pp on SkillsBench/SWE-Skills-Bench. REPAIRGATE recovers $+13.3/+17.1$ pp on SkillsBench/SWE-Skills-Bench.

Configuration	%Comp	#Skills	DriftRate
REPAIRGATE (full)	**60.24**	**1.4**	**2.4%**
w/o Gate head	50.60	1.7	6.0%
Gate-only on +Iter	56.02	1.05	1.2%
w/o Counterfactual	51.81	1.5	4.8%
w/o Length reg	58.43	2.9	2.4%
w/o Overlap reg	57.23	3.4	3.0%
SFT only (no GRPO)	55.42	2.6	4.2%

Table 4: Component ablation on Opus 4.6. #Skills: library size at family end. DriftRate: fraction of 166 trials with $R_{\text{deployed}} < R_{\text{without}}$ (decomposed in §4.4). "Gate-only on +Iter" filters the candidate patcher output and forces REPAIR to fall back to APPLY; it is a diagnostic overlay, not the full transition in Eq. (2).

cumulative skill counts.

4.5 Transfer to External Benchmarks

We evaluate our default REPAIRGATE checkpoint (no per-benchmark fine-tuning, no exposure to either external benchmark during training) on **SkillsBench** (Li et al., 2026) and **SWE-Skills-Bench** (Han et al., 2026) across the four Claude base agents from Table 1 (Table 3). We distinguish two iter modes: **+Iter (static)** injects a pre-curated skill (no evolution), and **+Iter (evolve)** lets an Opus 4.5 patcher rewrite after each trial. REPAIRGATE is held fixed as a drop-in maintainer on both.

We present tasks in benchmark-provided order and let the library accumulate across trials (no per-task reset). Static skills come from each benchmark's reference documents; the LLM-evolved iter uses an Opus 4.5 patcher with the same verifier feed as REPAIRGATE.

Findings. Static skills help only when aligned with the task ($+1.8/-0.10$ pp on SWE-Skills-Bench/SkillsBench), while LLM-evolved iter regresses by 5–9 pp—the same regression SKILLFLOW reports for weak base agents, now on benchmarks without shared family structure. REPAIRGATE, despite no training exposure to either benchmark, recovers $+13.3/+17.1$ pp over LLM-evolved iter (Table 3). Since the evolved iter uses a fixed Opus 4.5 patcher on both, these gains reflect REPAIRGATE mitigating harmful evolution under that fixed patcher rather than benchmark-native generalization. A per-domain breakdown on SKILLFLOW appears in Appendix K.

Failure boundaries. Residual failures concentrate in PPT-Formatting-Optimization and Sales-Pivot-Analysis, where REPAIRGATE achieves $1/8$ on each. In both families the gate often fires REPAIR, but the structured field-level protocol cannot express the required spatial coordinate transforms or pandas-AST rewrites. SPLIT_SKILL and MERGE_SKILLS are advisory-only (not executed by our field-level applier); new skill creation therefore enters only through accepting candidate upserts. Detailed cases and proposed AST/spatial extensions are in Appendix L.

5 Conclusion

Existing skill-library systems typically leave maintenance to a prompted LLM editor. We presented REPAIRGATE, an RL-trained post-trial controller that accepts, blocks, or overrides each candidate patch under a structured repair protocol. Across six base agents on SKILLFLOW, REPAIRGATE delivers $+7.73/+9.13$ pp over Vanilla/+Iter, with the largest gains on the weakest agents where prompted iteration regresses. Ablations attribute most of this gain to learned post-trial routing rather than free-form repair, and library growth is curbed mainly by abstention and overlap regularization at a standardized \$0.033/trial local maintainer overhead.

Limitations

(L1) Workflow-task focus. SKILLFLOW, SkillsBench, and SWE-Skills-Bench all exercise office and software workflows; embodied or open-domain dialog settings are out of scope. The structured repair protocol makes minimal task assumptions, so we expect the gate/repair design to extend, but a direct empirical test is left to future work.

(L2) Cost numbers depend on provider mix. Per-trial USD costs reflect Anthropic direct pricing for Claude and third-party aggregator pricing for MiniMax/GLM at measurement time. REPAIRGATE's $0.033/trial local wrapper overhead and $0.085/repair inference are grounded in A800 rental at $1.5/h and robust to API-side pricing changes.

(L3) Skill-mechanism overlap between training pool and evaluation. Although no SKILLFLOW benchmark trial appears in any SFT sample or GRPO prompt (§3.6, Appendix E), the training pool and evaluation share *workflow patterns*: the D1/D2/D3 mechanisms the repair head's gold edits target are by design the same failure mechanisms the held-out drift cases exhibit.

Use of AI Assistants

AI assistants were used in the preparation of this paper for writing polish and coding assistance. All research ideas, experimental designs, and conclusions are solely the work of the authors.

Ethics Statement

REPAIRGATE is a research artifact for studying skill-library maintenance in autonomous agents. We see no direct dual-use risk beyond what is already inherent in agent-orchestration code: a misconfigured library could accumulate harmful procedures, and a wrong abstention could leave a harmful skill unrepaired or suppress a useful update. The asymmetric reward (-2 false-apply, -0.5 false-abstain) biases toward abstention in ambiguous cases, and the structured repair protocol is auditable per action. Training follows the SKILLFLOW protocol but uses an independently authored auxiliary corpus; no SKILLFLOW evaluation instance appears in training data. The corpus contains no personal data; code, dataset, and checkpoint are released under Apache 2.0 (URL in abstract).

References

Anthropic. 2025. Claude code. https://github.com/anthropics/claude-code. Accessed: 2026-05-01.

Anthropic. 2026. Skills. https://github.com/anthropics/skills. Accessed: 2026-05-01.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495.

Boyu Gou, Demi Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *International Conference on Learning Representations*, volume 2025, pages 30851–30883.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, and 1 others. 2024. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pages 57734–57811.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. 2026. Swe-skills-bench: Do agent skills actually help in real-world software engineering? *arXiv preprint arXiv:2603.15401*.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Jian Hu, Xibin Wu, Zilin Zhu, Weixun Wang, Dehao Zhang, Yu Cao, and 1 others. 2024. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 6.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pages 32808–32824.

Jannik Kossen, Cătălina Cangea, Eszter Vértes, Andrew Jaegle, Viorica Patraucean, Ira Ktena, Nenad Tomasev, and Danielle Belgrave. 2022. Active acquisition

for multimodal temporal data: a challenging decision-making task. *arXiv preprint arXiv:2211.05039*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, and 1 others. 2026. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. 2025. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. 2026. Skillclaw: Let skills evolve collectively with agentic evolver. *arXiv preprint arXiv:2604.08377*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Xiaoxi Jiang, and Guanjun Jiang. 2026. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*.

Kolby Nottingham, Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Sameer Singh, Peter Clark, and Roy Fox. 2024. Skill set optimization: Reinforcing language model behavior via transferable skills. *arXiv preprint arXiv:2402.03244*.

OpenAI. 2025. Codex cli: Lightweight coding agent that runs in your terminal. https://github.com/openai/codex. Accessed: 2026-05-01.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, and 1 others. 2025. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pages 79791–79821.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

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

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, and 1 others. 2025. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, and 1 others. 2025. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7920–7939.

Yaxiong Wu and Yongyue Zhang. 2026. Agent skills from the perspective of procedural memory: A survey. *Authorea Preprints*.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, and 1 others. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

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

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, and 1 others. 2026b. Skillflow: Benchmarking lifelong skill discovery and evolution for autonomous agents. *arXiv preprint arXiv:2604.17308*.

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

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and 1 others. 2025. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

A Action Vocabulary Semantics

Operation	Semantics
ADD_PRECONDITION	Add a precondition.
TIGHTEN_WHEN_TO_APPLY	Narrow the applicability clause.
ADD_COUNTEREXAMPLE	Add an anti-pattern / counter-example.
SPLIT_SKILL	Advisory label for splitting an over-broad skill; not executed by our field-level applier.
MERGE_SKILLS	Advisory label for merging overlapping skills; not executed by our field-level applier.
DELETE_CLAUSE	Remove a harmful clause.
NOOP	Leave unchanged.

Table 5: Structured repair protocol used by the repair head. Five labels are executable field-level edits in our applier; SPLIT_SKILL and MERGE_SKILLS are high-level advisory labels and are not executed as card creation or merging in our experiments.

B Drift Analysis (Quantitative)

The 10 live drift instances introduced in §3.2 (per-family counts in Figure 5) involve the following skills: `excel-processing` (3 cases in Inventory-&-Finance-Integration), `edit-embedded-excel-in-pptx` (2 in Embedded-Data-Repair), `production-scheduling` (2 in Operational-Recovery-Planning), and one each of `excel-word-reporting`, `hwpx-editing`, and `image-to-excel-extraction`. These 10 cases are *held out* from REPAIRGATE's SFT supervision and used only for evaluation: on the held-out set, REPAIRGATE's gate head predicts REPAIR on 9 out of 10 drift positions (recall 0.90 at false-apply rate 0.13).

C Case Study: Drift on Neonatal Sepsis Harmonization

An Opus 4.5 agent iteratively refined a single `lab-data-harmonization` skill through T_1–T_8

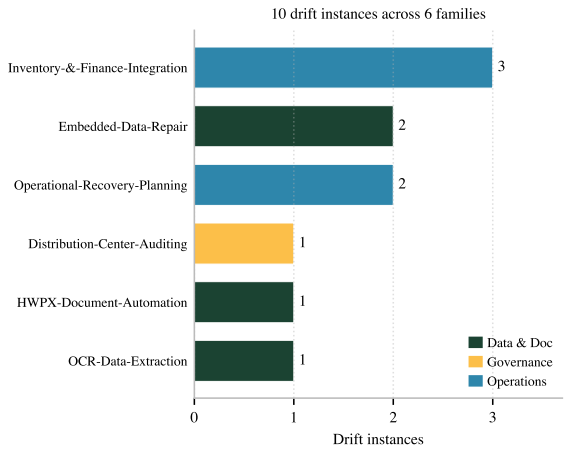

Figure 5: Drift instances per family under Opus 4.5 teacher, coloured by domain. 9 of 10 cases concentrate in Data & Doc and Operations.

in Medical-Data-Standardization. The mature skill grew from 0 to 16,979 tokens encoding adult clinical reference ranges, unit conversions, and verification procedures. We then re-ran T_9 (`neonatal-sepsis-harmonization`) loading the mature skill from a separate template directory; Table 6 compares the two runs.

	Original T_9 run	With mature skill
pytest pass	11/18 (61%)	2/18 (11%)
Wall time	179s	1,215s
Cost	\$0.42	\$0.56
Reward	0	0

Table 6: Drift on T_9 `neonatal-sepsis-harmonization`: loading the T_8-mature `lab-data-harmonization` skill drops per-assertion pass rate from 61% to 11% with agent and task identical.

The mature skill encoded adult-clinical heuristics that the agent then *confidently* applied to neonatal data, producing 16 assertion failures vs 7 originally. Binary task reward is identical (0); only fine-grained pytest counts reveal the regression. This is the textbook drift pattern that motivates REPAIRGATE's gate head and counterfactual reward.

D Detailed Drift Case Studies

This appendix expands the three drift mechanisms from §3.2 into worked examples. All three boxes are real teacher-run trajectories from our Opus 4.5 166-trial run. D2 (calc-engine, Embedded-Data-Repair) and D3 (logic-specialization, Distribution-Center-Auditing) are members of the 10 canonical drift instances(binary $R_{\text{with}} < R_{\text{without}}$). D1 (format-substring, Healthcare-Cost-Benefit) is a real failure pattern observed in our run but did *not* make the canonical 10 list because the binary verifier reward happened to satisfy $R_{\text{with}} = R_{\text{without}}$ (both runs failed the substring assertion at the binary level), even though per-assertion pytest counts—the same finer-grained signal that the Appendix C case study uses—reveal the skill-induced regression. We include D1 because the currency-format mechanism is a documented and reproducible failure mode worth illustrating, but it is excluded from the held-out repair-head evaluation pool of §4.4. For each pattern we show: (a) the trial that succeeded and seeded the skill, (b) the trial that inherited the skill and failed, (c) the verifier output, and (d) the REPAIRGATE action sequence that would repair it. All artifacts are released as part of our supplementary dataset.

Seeding trial. `harbor_mailerfill_45v90`, the family's first trial, succeeded ($R = 1$). It produced a ∼130-line skill `pharmacy-margin-analysis` whose "Output Format Validation" section taught: `"$72766.88"` → wrong (in JSON), `72766.88` → right. The skill body silently extrapolated the same currency style to summary text because mailerfill's reference summary used `"$X,XXX.XX USD"` formatting and the test passed.

Failing inheritor. `harbor_oncocooler_10v20`, run with the mature skill loaded ($R = 0$). Verifier output (verbatim):

```
AssertionError: assert '-79,751.70'
in 'Total 10-day annual margin:
-$79,751.70 USD\n...'
at /tests/test_outputs.py:165 in
test_summary_requirements
```

The test computes `f'{value:,.2f}'` which gives `-79,751.70`. The agent's text contains `-$79,751.70` so the substring check fails. *The agent's mathematical answer is exactly correct*; the only divergence is the `$` character between the sign and the digits. The same failure recurs on `harbor_reagentkit_bulk`, `harbor_syncpack_28v56`, and `harbor_vaxcrate_6v12`—all four downstream trials in this family that inherited the skill.

Why T_1 passed. Mailerfill's margins were positive: `"$32,408.40"`. The pattern `32,408.40` *is* a substring of `$32,408.40`, so the same template that breaks T2–T5 satisfied T1.

Repair.

```
<gate>REPAIR</gate>
<edit>
  ACTION: ADD_COUNTEREXAMPLE
  PATCH: "Negative margins must use
  plain
       '-X,XXX.XX' format; '$' between
       sign and digits breaks substring
          assertions."
  ACTION: TIGHTEN_WHEN_TO_APPLY
 PATCH: "tested only on positive-margin
       tasks; negative outputs require
          alternate format."
</edit>
```

Seeding trial. `buffer-dilution-matrix-repair` succeeded with skill `edit-embedded-excel-in-pptx`, which teaches: "PowerPoint files are ZIP archives. Embedded Excel workbooks live at `ppt/embeddings/Microsoft_Excel_Worksheet.xlsx`. The embedded xlsx is itself a ZIP containing `xl/worksheets/sheetN.xml`." The agent edits the XML directly to set cell values.

Failing inheritor. `fx-cross-rate-inverse-fix`: the workbook contains formula cells (`=1/B2` for the inverse rate). Verifier output (verbatim):

```
wb =
load_workbook_from_pptx(RESULT_FILE,
data_only=True)
direct_value =
ws.cell(*direct_coord).value
> assert direct_value is not None
AssertionError: Direct rate cell
is empty (no cached value)
```

The XML edit set the formula string but did not refresh the cached evaluation; `data_only=True` reads cached values, finds `None`, and rejects. The same root cause recurs on `catalyst-balance-matrix-sync`, the second of the two Embedded-Data-Repair drift instances in our run.

Repair.

```
<gate>REPAIR</gate>
<edit>
  ACTION: ADD_PRECONDITION
 PATCH: "If any target cell contains a
       formula, the XML edit alone is
       insufficient. After re-zipping,
          run `libreoffice --headless
           --calc
        --convert-to xlsx` to refresh
       cached evaluations before saving
          back into the pptx."
</edit>
```

A more aggressive library-level diagnosis is `SPLIT_SKILL` into two skills (literal-cell vs formula-cell variants). In our applier this is logged only as an advisory label; the executable repair remains the field-level `ADD_PRECONDITION` patch above.

Seeding trials. Three early trials—`harbor_outbound_manifest_audit`, `harbor_promo_register_audit`, `harbor_receiving_exception_audit`—all succeeded with skill `excel-word-reporting`. The skill's `description` field grew over the three trials to enumerate ten audit sub-cases: "SLA audits, threshold-based error detection, compliance checks, lookup-table-driven calculations, alias normalization, planned-vs-actual disposition audits, outbound manifest audits, scan log validation, zone mismatch detection, cycle-count variance audits." This is library inflation. The skill body, however, encodes a single concrete flag formula: `flag = 1 if value > threshold else 0`.

Failing inheritor. `harbor_returns_disposition_audit`: returns disposition is not threshold-based; it requires looking up `latest_completed` state and an `alias_map`. Verifier output:

```
> assert int(out[8]) ==
expected_missing
AssertionError: assert 0 == 1
```

The agent applied the threshold formula because the description claimed disposition audits were in scope; the body's threshold

formula returned 0 where the correct stateful logic would return 1.

Repair.

```
<gate>REPAIR</gate>
<edit>
  ACTION: DELETE_CLAUSE
  PATCH: "Remove sub-case enumeration
  from
         description that body cannot
         actually solve (specifically
       'planned-vs-actual disposition
          audits' and 'cycle-count
          variance audits')."
  ACTION: ADD_COUNTEREXAMPLE
  PATCH: "returns-disposition uses
          latest_completed lookup and
        alias_map, not threshold-flag
         logic; the threshold formula
          here is wrong."
</edit>
```

Summary across the three patterns. Each drift type maps onto a counterfactual signal that does not require solving the task and onto a structured edit that does not require rewriting the skill from scratch. The main text (§3.2) explains why this is exactly the regime where a small trained policy can outperform a frontier prompted teacher: the meta-task of recognizing scope mismatch and emitting the corresponding structured action has both a clear training signal (counterfactual Δ_t^{cf}) and a small, learnable output space (7 actions $\times$ short patch text).

E Training Corpus Construction

This appendix details the construction of the SFT and GRPO training data. We use SKILLFLOW's *protocol* (natural-language tasks, Docker, pytest verifier, evolving `SKILL.md`) but no SKILLFLOW *instances*: every task description, Docker environment, verifier, trajectory, skill snapshot, and teacher patch in our training pool is independently authored.

E.1 Auxiliary Lifelong-Learning Corpus

We construct an auxiliary corpus of four workflow families with eight tasks each, structurally analogous to SKILLFLOW's domains but with all artefacts independently authored:

- *Tabular cleaning* — CSV/Excel-like data normalization, grouping, aggregation; exercises format-substring drift.
- *Document transformation* — Markdown / JSON / YAML extraction and conversion; exercises format-assumption drift.
- *Stateful operations* — ticketing, inventory updates, multi-step approval workflows; exercises early-task logic over-specialization.
- *Synthetic finance* — invoice reconciliation, tax computation, decimal arithmetic with synthetic data; exercises calculation-engine drift.

Each task carries an independently authored description, Docker environment, pytest verifier, binary reward, and reproducible input data. Within each family, task difficulty increases monotonically and the skill library resets between families, mirroring SKILLFLOW's protocol.

E.2 Pilot Run on the Auxiliary Corpus

We run an Opus 4.5 lifelong-learning pilot on the 32 auxiliary tasks. For each task T_t^{aux} we collect: (i) the with-skill rollout $(\tau_t^{\text{with}}, R_t^{\text{with}})$ under the current snapshot $\mathcal{S}_{t-1}^{\text{aux}}$; (ii) a no-skill counterfactual $(\tau_t^{\text{without}}, R_t^{\text{without}})$ on the same T_t^{aux} with an empty library; (iii) the teacher patch Δ_t^{aux} producing the next snapshot $\mathcal{S}_t^{\text{aux}}$. The counterfactual gain $\Delta_t^{\text{cf}} = R_t^{\text{with}} - R_t^{\text{without}}$ provides automatic gate labels for the pilot.

E.3 Skill-Defect Injection

The pilot alone yields too few negative-transfer events. We additionally inject controlled defects into auxiliary skill snapshots, instantiated as a corrupted skill $\mathcal{S}_{\text{bad}}$ paired with a canonical fixed skill $\mathcal{S}_{\text{fixed}}$; the difference is mapped to a 7-action gold edit sequence. Defect categories include the three drift mechanisms in Appendix D (D1 format-substring, D2 calc-engine, D3 logic-specialization) and edge cases (hard-coded paths, hard-coded column names, missing unit conversion, duplicate skills that should be merged, mixed skills that should be split, harmful clauses that should be deleted, plus clean-skill prompts where NOOP/ABSTAIN is the gold action). Each corrupted snapshot is rerun on its auxiliary task to obtain a real trajectory and verifier output.

E.4 SFT Corpus ($\sim$1,000 items)

The SFT corpus contains $\sim$500 gate items + $\sim$500 repair items, both derived from §E.2–E.3. All SFT

prompts include the candidate patch Δ_t^{cand} in addition to the skill snapshot, task description, with-skill trajectory, and verifier output, matching the deployment input in §3.1.

Gate items ($\sim$500). Distribution APPLY:ABSTAIN:REPAIR $\approx$ 32:38:30. Sources: (a) per-task counterfactual labels from the pilot (Δ_t^{cf}); (b) cross-family ABSTAIN pairs constructed by routing a mature skill from one auxiliary family to a structurally unrelated family task; (c) intermediate-snapshot pairs from the pilot, where an early snapshot is paired with a later task to elicit ABSTAIN/REPAIR when the older snapshot has become drift-prone.

Repair items ($\sim$500). Coverage: D1 ($\sim$125), D2 ($\sim$125), D3 ($\sim$125), mixed defects ($\sim$75), and edge-case actions (SPLIT_SKILL/MERGE_SKILLS/DELETE_CLAUSE/NOOP, $\sim$50). Gold action sequences are produced by mapping the $\mathcal{S}_{\text{bad}} \rightarrow \mathcal{S}_{\text{fixed}}$ diff onto the structured repair protocol, then validated by re-applying the patch and verifying the auxiliary task passes. *We do not* treat the raw teacher patch as gold; pilot patches are used only as weak supervision and are filtered for unambiguous mappings. When SPLIT_SKILL or MERGE_SKILLS appears in training annotations, it is treated as a high-level advisory label for attribution; the environment reward is determined by the resulting field-level library state after applying the executable edits.

E.5 GRPO Pool ($\sim$500 prompts)

The GRPO pool draws from the same auxiliary distribution but is prompt-disjoint from the SFT validation split. Each prompt is ChatML-formatted and contains the skill snapshot, task description, with-skill trajectory, verifier output, and the candidate patch Δ_t^{cand}. For each auxiliary prompt, Δ_t^{cand} is precomputed by the standard candidate patcher and held fixed across the $G = 8$ GRPO samples; A_ϕ^{train} is used only to evaluate the resulting library state. Prompts are selected to satisfy: (i) Docker environment runs deterministically; (ii) policy-emitted edits can be mechanically applied to `SKILL.md`; (iii) verifier reward distinguishes good from bad edits with low variance; (iv) per-prompt rollout time is bounded.

Composition (approximate): repairable drift prompts ($\sim$170 across D1/D2/D3 + mixed); cross-family or under-evidenced prompts that should ABSTAIN ($\sim$160); positive-transfer prompts that should APPLY ($\sim$130); and ambiguous boundary cases ($\sim$40). The label distribution after augmentation is approximately APPLY 32%, ABSTAIN 38%, REPAIR 30%. We hold out $\sim$50 of these prompts at the family level for GRPO validation.

E.6 Disjointness Audit

To verify no SKILLFLOW artefact leaks into training, we ran three checks against the 166 SKILLFLOW evaluation trials: (i) an exact-string scan over task descriptions and verifier code (zero matches); (ii) a 5-gram overlap scan with threshold ≥ 10 shared 5-grams (zero matches above threshold); (iii) a manual review of the top-20 embedding nearest-neighbours from each SKILLFLOW task to our training prompts. Each training item carries a provenance manifest recording its workflow family, base task index, defect type, and split membership, together with content hashes for the skill snapshot, task description, and verifier.

F Training Setup and Wall-Clock Timing

We measured every training step on the actual hardware used for the experiments reported in the main text: 2×NVIDIA A800-SXM4-80GB on autodl, CUDA 12.8, PyTorch 2.10.0+cu128, flash-attn 2.8.1 (sm_80 prebuilt wheel), Python 3.12.

F.1 Phase 1: Supervised Fine-Tuning (LlamaFactory)

Data and model. The SFT corpus of $\sim$1,000 samples (§3.6) is split $\sim$900/100 into train and val (val draws roughly equally from the gate and repair pools), packed in ChatML message-list format with a 8192-token cutoff and sequence packing disabled to avoid mixing gate and repair samples. The base is Qwen3.5-9B-Instruct; LoRA is rank 32, alpha 64, dropout 0.05, applied to all linear modules (86.6M trainable params, 0.91% of 9.5B). Precision is bf16 with flash-attn auto-dispatch, and gradient checkpointing is enabled.

Optimization. AdamW at LR 1×10^{-4}, cosine scheduler, warmup ratio 0.03. Per-device train batch is 1 with gradient accumulation 8 (effective batch 8). We train for 2 epochs over the $\sim$900-sample training split.

Wall-clock. SFT completes in $\sim$1–2 hours end-to-end on a single A800-80G.

F.2 Phase 2: GRPO (veRL)

This subsection records GRPO implementation parameters; the overall objective is summarized in §3.6.

Data. $\sim$500 train prompts and $\sim$50 val prompts (held out at the family level), constructed per Appendix E. Each prompt is ChatML `[system, user]` containing the skill markdown, task description, agent trajectory, verifier output, and the precomputed candidate patch Δ_t^{cand}, truncated at the parquet stage to fit a max-prompt length of 12,288 tokens. The custom reward function implements §3.5; per-prompt response sampling uses $n = 8$ candidate (gate, edit) tuples for group-relative advantage estimation.

Model and engine. The cold-start initialization is the SFT LoRA adapter, loaded but not merged into the base. KL coefficient is 0.01 (low-variance estimator), entropy coefficient 0.001. The reference model uses CPU param offload to fit alongside the actor and the vLLM rollout; FSDP at bf16 with gradient checkpointing.

Rollout (vLLM). TP size 2 (both A800 GPUs), $n = 8$ candidates per prompt.

Optimization. LR 5×10^{-6}, 5 epochs over the augmented training set, effective batch 8, max token length 16,384 per GPU. With 500 prompts and batch 8, this is $\sim$63 outer GRPO steps per epoch and $\sim$315 outer steps in total.

Wall-clock and resource use. Both A800s sit at 90–95% utilization during a step; steady-state GPU memory is $\sim$37 GB / 80 GB per card. Each outer GRPO step interleaves vLLM rollout (which dispatches all 64 candidate edits to a 16-way parallel Docker rerun pool and gathers verifier outcomes), reference-model log-prob computation, and the PPO update. The full GRPO phase runs 5 epochs over the $\sim$500-prompt training set, producing $\sim$20,000 environment-grounded rollouts in $\sim$52 h on 2×A800-80G.

Optimization and GRPO rerun cost. GPU compute for LoRA optimization is \$156 on 2×A800-80G at \$1.50/h (52 h × 2 GPUs at 16-way Docker parallelism). Training-time base-agent API cost for the GRPO Docker reruns is **\$**1,080, logged across the $\sim$20,000 rollouts of one full training run ($\sim$\$0.054/trial on average). The LoRA-optimization and GRPO environment-rerun cost is therefore $\sim$**\$**1,238, *excluding* the one-time auxiliary-corpus generation with Opus 4.5 (§E.2). All reported deployments amortize this single training run.

Implementation notes. We record the non-default settings required to reproduce the reported GRPO runs. We set `max_prompt_length=12288` and truncate prompts to at most 11K tokens during parquet construction, since the default length limit silently drops long trajectory-conditioned skill-evolution prompts. To fit actor, reference, and rollout workers on 2×A800-80G, we enable actor/reference parameter offload, reduce the veRL default batch sizes, and set vLLM `gpu_memory_utilization=0.45`. API endpoint variables are left unset unless explicitly overridden by a valid URL; empty-string endpoints can cause SDK connection failures. Exact configuration files are included in the released artifact.

G Inference and Training Cost

Unified cost framing. We report costs under the same SKILLFLOW logging convention across methods and use the REPAIRGATE-local cost as the standardized incremental quantity. Since +Iter and +REPAIRGATE share the same candidate patcher, our main cost claim is relative to +Iter: REPAIRGATE adds a standardized \$0.033/trial local A800 wrapper overhead. We do not rely on a fine-grained decomposition of base-agent and patcher API cost in the main paper. The released artifact patches the original SKILLFLOW LiteLLM wrapper to record patcher-side token usage in future runs for reproducibility.

Per-trial RG-local overhead (\$0.033). The gate head runs on every trial ($\sim$20 s of 9B-LoRA inference, \$0.008). The repair head emits a structured repair proposal only when the gate predicts REPAIR; otherwise REPAIRGATE either accepts or rejects Δ_t^{cand}. From §4.4 this occurs on $49/166 = 29.5\%$ of trials, contributing $0.295 \times \$0.085 \approx \$0.025$. For cost reporting, we use the observed Opus 4.6 decision distribution as a standardized local-overhead estimate; actual local overhead scales linearly with the per-base-agent repair-call rate. Sum $\approx$\$0.033.

Per-repair inference (\$0.085). Cost of one full repair-proposal call (longer generation: action sequence plus patch text). Compared with an Opus-class free-form patch call (estimated at $\sim$\$0.4 per

Family	Domain	+Iter	+REPAIRGATE	Δ
Cross-Format-Data-Reconciliation	Data & Doc	8/8	8/8	0
Document-Fraud-Detection	Governance	8/8	8/8	0
Industry-Correlation-Analysis	Finance	8/8	8/8	0
Production-Capacity-Planning	Operations	8/9	8/9	0
Supply-Chain-Replenishment	Operations	7/9	8/9	+1
Operational-Recovery-Planning	Operations	6/8	5/8	−1
HWPX-Document-Automation	Data & Doc	7/8	8/8	+1
Distribution-Center-Auditing	Governance	7/8	7/8	0
OCR-Data-Extraction	Data & Doc	6/8	7/8	+1
Inventory-&-Finance-Integration	Operations	5/8	6/8	+1
Embedded-Data-Repair	Data & Doc	6/8	6/8	0
SEC-13F-Financial-Analysis	Finance	3/8	3/8	0
Medical-Data-Standardization	Healthcare	6/9	6/9	0
Healthcare-Cost-Benefit-Analysis	Healthcare	2/9	3/9	+1
Compensation-Scenario-Modeling	Governance	1/8	1/8	0
DMAIC-Quality-Analysis	Operations	1/9	2/9	+1
Financial-Statement-Rolling	Finance	1/9	2/9	+1
PPT-Formatting-Optimization	Data & Doc	0/8	1/8	+1
Sales-Pivot-Analysis	Data & Doc	0/8	1/8	+1
Weighted-Risk-Assessment	Data & Doc	1/8	2/8	+1
Overall		**91/166 (54.82)**	**100/166 (60.24)**	**+9**

Table 7: Per-family completion on Opus 4.6 under +Iter and +REPAIRGATE (166 trials). Aggregates $91/166 = 54.82\%$ and $100/166 = 60.24\%$ match Table 1.

substantive patch at Opus pricing), a single structured repair-proposal call by REPAIRGATE is several times cheaper on repair-side inference.

Optimization and GRPO rerun cost. Training-side compute is \$1.50 for SFT, \$156 for GRPO LoRA optimization (52 h × 2 A800 at \$1.50/h, 16-way Docker parallel), and \$1,080 for base-agent API calls logged across the GRPO Docker reruns—totalling ∼\$1,238 for one REPAIRGATE-v1 optimization run, excluding the one-time auxiliary-corpus generation with Opus 4.5 (§E.2). All deployments and benchmarks reported in the main paper amortize this single run.

H Hyperparameters

LoRA rank=32, alpha=64, target=all linear modules, dropout=0.05.
SFT: LR 1×10^{-4}, effective batch 8, 2 epochs, ∼1 h on 1×A800-80G.
GRPO: $G = 8$ rollouts, KL coefficient 0.01, clip ratio $[0.2, 0.28]$, entropy coefficient 0.001, LR 5×10^{-6}, 5 epochs over ∼500 augmented prompts (∼315 outer steps), ∼52 h on 2×A800-80G (16-way Docker parallel).
Reward weights: $\alpha = 0.05$ (length), $\beta = 0.10$ (overlap), $\gamma = 0.5$ (gate); asymmetric gate penalties $\{+1, -2, -0.5, -0.3\}$.
Inference: vLLM, temperature 0.0 for evaluation.

I Reward-Coefficient Sensitivity

To verify that the main results do not depend on a knife's-edge choice of (α, β, γ), we sweep each coefficient at $\times0.5$ and $\times2.0$ around the default. To save compute, every sweep run uses the same GRPO configuration as the main results but trains on a 20% random subset of the ∼500 GRPO prompts (otherwise the 7 runs would cost ∼360h on 2×A800-80G); a calibration row at the default weights on the same 20% subset is included as the reference point. Evaluation is on the full 166 SKILLFLOW trials with Claude Opus 4.6 as the base agent.

Setting	(α, β, γ)	%Comp
Default (20%-data calibration)	$(0.05, 0.10, 0.5)$	56.63
$\alpha\times0.5$	$(0.025, 0.10, 0.5)$	56.63
$\alpha\times2.0$	$(0.10, 0.10, 0.5)$	56.02
$\beta\times0.5$	$(0.05, 0.05, 0.5)$	56.02
$\beta\times2.0$	$(0.05, 0.20, 0.5)$	56.02
$\gamma\times0.5$	$(0.05, 0.10, 0.25)$	55.42
$\gamma\times2.0$	$(0.05, 0.10, 1.0)$	56.02

Table 8: Reward-coefficient sensitivity on Claude Opus 4.6 (20% GRPO data, full 166-trial evaluation). All six perturbations stay within −1.21 pp of the calibrated default; the gate-weight γ accounts for the largest swing (max −1.21 pp at $\times0.5$), the length-weight α the smallest (max −0.61 pp at $\times2.0$), consistent with the component hierarchy of Table 4. None of the choices is on a knife's edge.

J Per-Family Results on Claude Opus 4.6

Table 7 reports per-family completion rates on Claude Opus 4.6 under prompted +Iter; this is the source data underlying the Opus 4.6 row of Table 1 (overall $91/166 = 54.82\%$) and the per-domain bars of Figure 3. Counts are pass / total trials within each family. We also list the corresponding +REPAIRGATE numbers in the rightmost column for reference; the Δ column highlights families where the gate/repair pipeline contributes the most absolute lift.

K Per-Domain Breakdown on SKILLFLOW

Figure 6 aggregates the per-family counts in Table 7 into the five SKILLFLOW domains, comparing Vanilla and REPAIRGATE on Opus 4.6. REPAIRGATE outperforms Vanilla on every domain, with the largest absolute lift on Data & Doc.

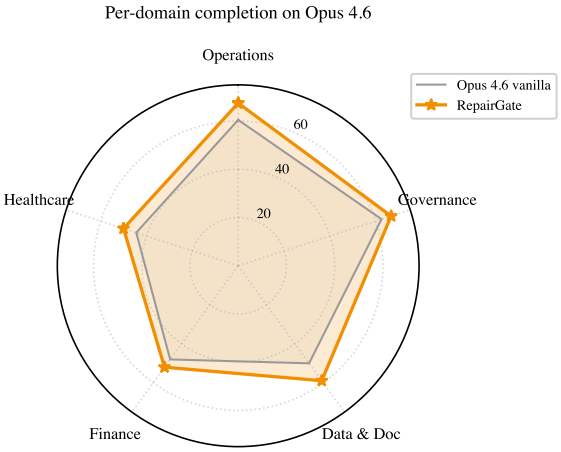

Figure 6: Per-domain completion on Opus 4.6 across five SKILLFLOW domains: Vanilla (no skill library) vs. REPAIRGATE.

L Failure Boundary Details

Spatial and code-AST drifts. PPT/UI tasks require coordinate transforms (e.g., "shift element A up 0.5 cm", "swap z-order of shapes B/C") that text-level actions like TIGHTEN_WHEN_TO_APPLY or ADD_PRECONDITION cannot express. In data-analysis tasks, skills encapsulate pandas `groupby` keys, measure stacking, and fill policies; when the target schema changes (e.g., `[region, quarter]`→`[region, quarter, channel]`), the repair requires AST-level rewriting rather than markdown text modifications. On PPT-Formatting-Optimization, the gate correctly fires REPAIR on 88% of trials, but the repair head defaults to defensive but insufficient TIGHTEN_WHEN_TO_APPLY-style actions. Extending the action vocabulary with AST-aware operators (e.g., REWRITE_GROUPBY, INJECT_FILLNA) and spatial-aware operators is the most direct path to closing this residual gap; the gate-head supervision carries over unchanged.