Repo2Skill-Evo: Repository Skills Go Stale in Silence

[1]ByteDance, [2]Peking University, [3]Beijing Jiaotong University

Full author list in Contributions

Abstract

Large language model (LLM) agents increasingly operate over evolving software repositories, where success depends on repository-specific procedural knowledge: which APIs to call, which scripts to run, and which conventions the current release expects. Agent skills externalize this knowledge into reusable units, and prior work shows that they can improve agent performance. What remains unclear is whether that improvement is durable. The same version specificity that makes a skill useful also makes it fragile: after a release, it may become stale without raising any explicit signal, while continuing to provide obsolete guidance. Externalizing knowledge into a skill can therefore make its decay invisible.

We study whether agents can keep this externalized knowledge current. Repo2Skill-Evo casts each release transition as a skill-maintenance task: given a V_1 skill set and the official $V_1 \to V_2$ patch, an agent must update obsolete skill content while preserving guidance that remains valid. Across 57 real-world repositories and 105 selected release transitions, every evaluated transition invalidates part of the V_1 skill set. Yet six frontier agents reach only 29.9%–69.7% avg@3 macro F_1 under a patch-grounded removal metric that balances stale-content recall against over-editing precision. Across runs, two opposing errors dominate: incomplete coverage of affected files in the skill set leaves stale content untouched, while overbroad editing is associated with higher recall but lower precision. **Repository skills go stale in silence, and even frontier agents cannot reliably maintain them.**

1 Introduction

Large language model (LLM) agents increasingly act through tools and interactive environments [19, 28, 36], and repository-level software engineering has become one of their most demanding settings. To complete realistic repository tasks, an agent must inspect code, read documentation, edit files, and run scripts in an evolving codebase [11, 35]. Success therefore depends not only on the underlying model, but also on repository-specific procedural knowledge: which APIs to call, which scripts to run, how modules are configured, and which conventions the current release expects. Extracting this knowledge from scratch for every task is costly and unreliable, since it requires cross-file retrieval and structural understanding that even strong agents do not consistently achieve [15, 18].

Agent skills offer a compact alternative. They externalize reusable procedural knowledge into on-demand units that agents can retrieve and follow [8, 10, 16]. Prior work shows that skills can improve downstream agent performance [8, 16, 41], and repository-local skills are beginning to appear beyond benchmark settings [1, 27]. Skills are thus becoming persistent agent-facing resources across tasks and releases.

We take this utility as our starting point and ask: once a repository evolves, can agents keep its skills from silently going stale? A repository skill is useful precisely because it encodes version-specific knowledge, but

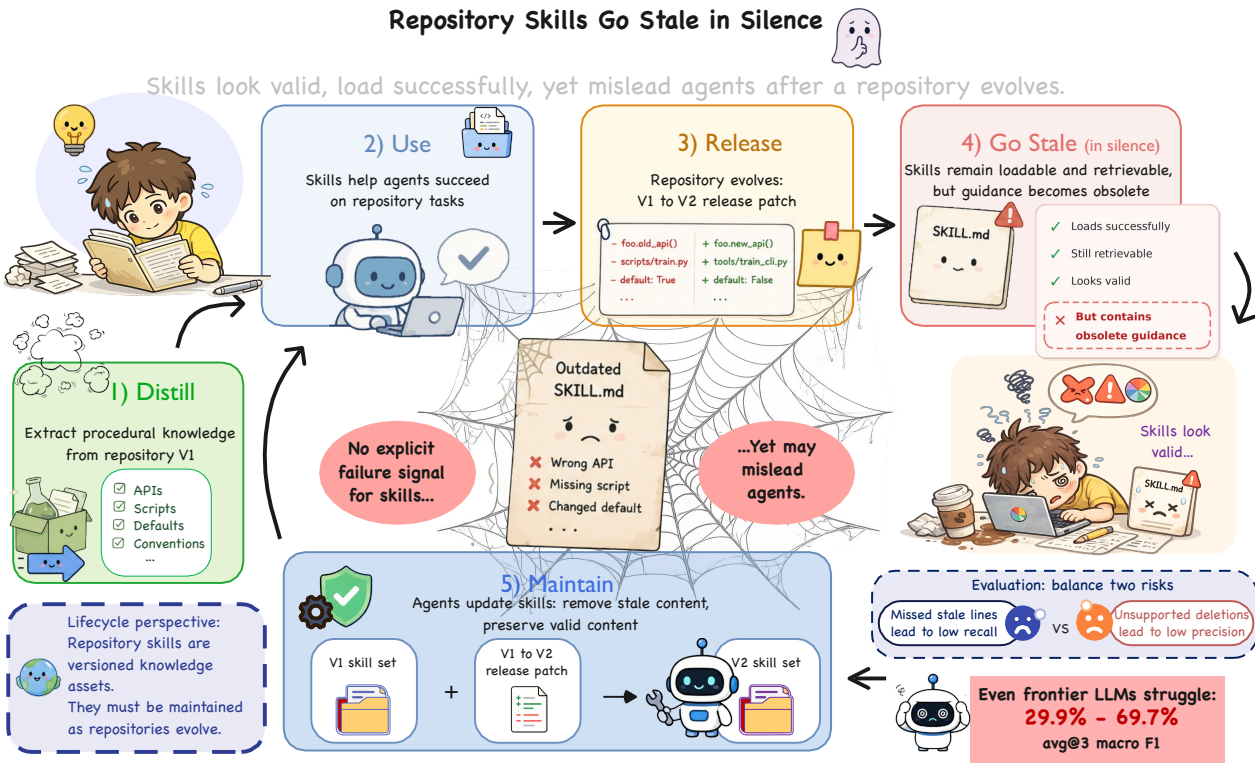

Figure 1 Lifecycle of repository skills and the Repo2Skill-Evo maintenance setting. Repo2Skill distills repository-specific procedural knowledge from a V_1 repository into a traceable skill set. After the repository evolves from V_1 to V_2, previously valid skill content may become silently stale: it remains loadable and retrievable while continuing to provide guidance that no longer matches the current repository state. Given the fixed V_1 skill set and the official release patch, a maintenance agent produces the updated skill set $\mathcal{S}_{V_2}$. Repo2Skill-Evo evaluates whether the agent removes stale content while preserving guidance that remains valid.

that same specificity ties its correctness to a particular repository state. After a release, a skill distilled from the previous version may remain loadable and retrievable while continuing to provide guidance that no longer matches the current repository state. Because no explicit signal marks this mismatch, the outdated guidance may continue to be retrieved and followed. We refer to this release-driven failure mode as *silent staleness*. Externalizing version-specific knowledge into a skill preserves its form while rendering its decay invisible. Figure 1 summarizes this lifecycle and the resulting maintenance setting.

We therefore treat repository-grounded skills as versioned knowledge assets and center our study on their maintenance. Although related to code migration, the maintained object is the agent-facing guidance derived from a repository, not the repository code itself. Because a release may invalidate only part of a skill package, maintenance requires removing or revising stale content while preserving what remains valid.

To study this problem at scale, Repo2Skill-Evo combines three components. First, Repo2Skill uses staged distillation and expert refinement to produce a fixed V_1 skill set. Second, given this skill set and the official $V_1 \to V_2$ release patch, a maintenance agent must remove or revise obsolete skill content while preserving guidance that remains valid. Third, a strictly patch-grounded headline metric rewards the removal or revision of patch-verified obsolete V_1 skill lines while penalizing edits to other V_1 skill lines, complemented by a rubric-based NL Judge that provides a five-dimensional assessment of the resulting V_2 skill set [12, 20, 40].

A focused utility study further motivates the maintenance problem. Across ten repositories randomly sampled from the corpus, skills yield the largest absolute gains on tasks with the lowest baseline utility. Within this sample, the pattern suggests that maintenance may matter most where agents have limited repository knowledge without external support.

Our main evaluation across 57 repositories and 105 selected release transitions yields two findings. **First, agents do not reliably maintain repository skills across releases.** Each selected transition includes a V_1 skill set with patch-verified stale content (median: 92 lines), yet six frontier agents achieve only 29.9%–69.7% avg@3 macro F_1. **Second, failures reflect complementary localization and edit-selection bottlenecks.** Incomplete coverage of affected skill files leaves stale content untouched, whereas broader edits are associated with higher recall but lower patch-grounded precision. Oracle skill-file localization on the 20 hardest transitions improves performance, but substantial residual errors remain, indicating that localization alone does not resolve within-file evidence tracing and edit selection.

This paper makes three contributions:

- We characterize **silent staleness**, a release-driven temporal failure mode of repository-grounded skills, and frame these skills as versioned knowledge assets.
- We introduce **Repo2Skill-Evo**, a patch-grounded maintenance benchmark covering 57 repositories and 105 official release transitions, with fixed V_1 skill sets and patch-verified obsolete-line annotations.
- We evaluate six frontier agents and identify two key maintenance bottlenecks: incomplete localization of affected skill files and imperfect edit selection.

2 Related Work

2.1 Repository-Level Agents and Software Evolution

Repository-level software engineering is a central setting for evaluating LLM agents. SWE-bench casts real GitHub issues as end-to-end repair tasks that require understanding a codebase and producing test-passing patches [11], while SWE-agent shows that the agent–computer interface shapes how agents navigate, edit, and test repositories [35]. Other systems study issue resolution, localization, and repair scaffolding [34, 39]. RepoBench emphasizes cross-file context [18], and RepoMirage shows that broad repository access does not necessarily yield structural understanding [15].

A parallel line studies software evolution directly, including library migration, API refactoring, and migration detection [2, 9, 13, 31]. Recent agent benchmarks extend from static repair to release-note-derived evolution tasks [32], continuous-integration loops [4], and chained release-level upgrades [14]. These works center on source code or downstream client usage. Repo2Skill-Evo instead maintains the agent-facing knowledge that describes how to use a repository. Because this knowledge is a separate artifact, its maintenance does not reduce to code migration.

2.2 Agent Skill Construction and Utility

Agent skills package reusable procedural knowledge into modular units that agents retrieve when relevant. Surveys organize this area around the skill lifecycle, spanning representation, acquisition, retrieval, and evolution [10, 42]. Skills can be generated from source corpora [41] and improved through execution and verifier feedback [21, 37]. Utility benchmarks report heterogeneous effects: curated skills can improve performance, whereas self-generated or mismatched skills do not reliably help [8, 16, 41].

Most of this work evaluates skill construction or use within a fixed environment snapshot. Repository-local skills also appear outside benchmarks. `SKILL.md` anchors an emerging cross-agent format [1], and PyTorch ships skills for dispatch-macro migration, Metal kernel implementation, AOTInductor debugging, and docstring maintenance [27]. These examples establish repository skills as practical agent-facing artifacts, but do not establish that they remain valid after their repositories advance to new releases.

2.3 Agent Skill Lifecycle and Maintenance

Software engineering has long recognized that natural-language artifacts can drift out of sync with code, as in code-comment inconsistency [33]. A repository skill serves as an agent-facing analogue carrying direct operational consequences. It is not merely reviewed by human maintainers but can also be retrieved and

executed by autonomous agents. Runtime feedback can support skill repair after an observed execution failure [23], but silent staleness may provide no error log to react to.

Recent work addresses distinct parts of the skill lifecycle. SkillGuard detects environment-contract violations and uses them to localize repair [5]. SkillOps and Library Drift tackle concerns surrounding skill libraries. SkillOps mitigates technical debt via typed contracts and graph-structured maintenance [26], while Library Drift investigates retrieval degradation stemming from unregulated expansion within self-evolving skill banks [38]. SkillHone focuses on iterative revision by preserving decision histories and evaluation evidence across sessions [17]. An empirical study complements these systems by characterizing how registry and personal-use skills are reused, adapted, and maintained in practice [6].

Repo2Skill-Evo isolates release-conditioned maintenance. The release change is provided as an official patch, and the challenge is to map its implications onto a fixed repository-grounded skill set while preserving unaffected guidance. Unlike skill internalization [22], the maintained object remains an external artifact whose validity is tied to a software release.

3 Repo2Skill-Evo: Maintaining Repository Skills across Releases

Repo2Skill-Evo has two components, a procedure for obtaining a fixed, traceable V_1 skill set to maintain, and a release-level task that asks whether agents can keep that set current. We describe the skill set first, then the maintenance task, scaffold, and evaluation. Figure 2 summarizes the full pipeline.

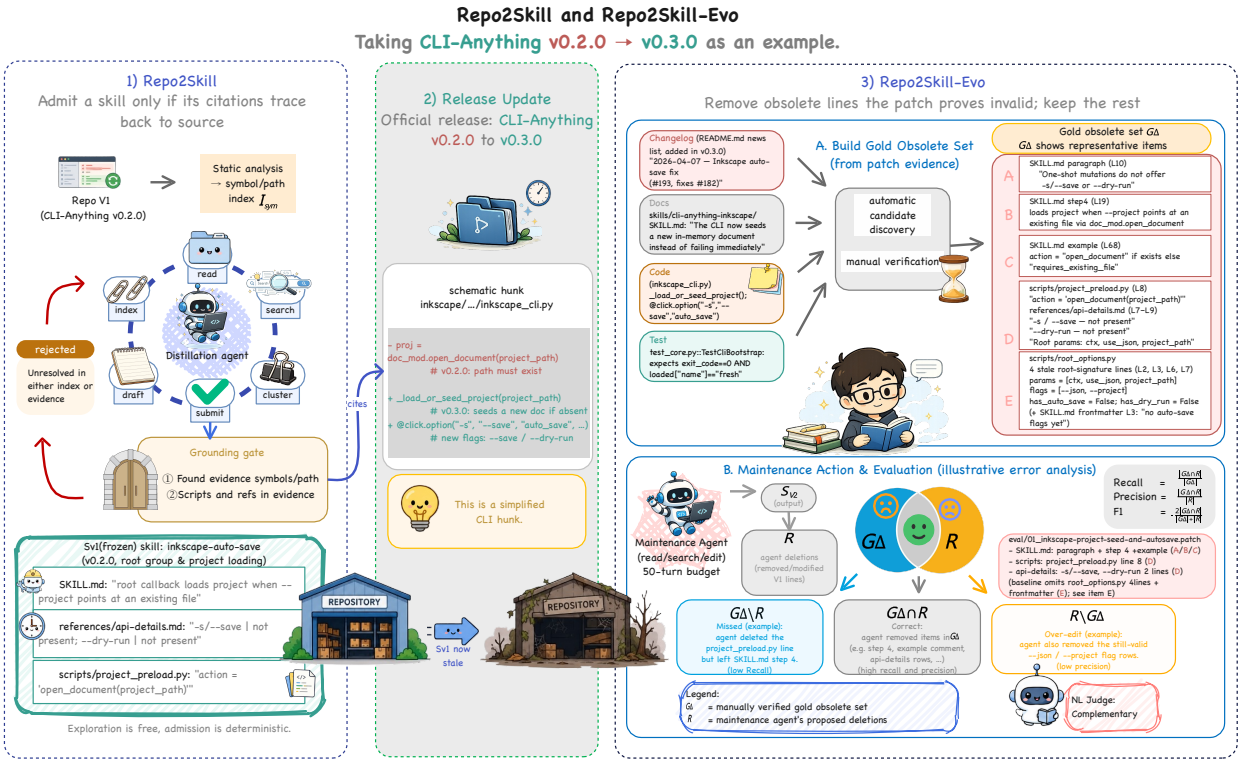

Figure 2 Repo2Skill and Repo2Skill-Evo on a release example. Using a simplified `CLI-Anything` $v0.2.0 \rightarrow v0.3.0$ update, the figure illustrates three stages: (1) Repo2Skill generates candidate V_1 skill packages, after which an expert selects, revises, and verifies one to produce the fixed V_1 skill set; (2) the release patch changes a documented schema behavior; and (3) Repo2Skill-Evo asks a maintenance agent to remove obsolete lines while preserving valid content. Evaluation uses the patch-grounded gold obsolete set G_Δ (Section 3.4). The same construction is applied across all 105 release transitions.

3.1 Repo2Skill: Distilling a Traceable Skill Set

Before asking whether agents can keep a skill set current, we first need a fixed V_1 skill set whose repository-specific claims can be traced to source evidence. Repo2Skill serves this role by distilling repository-specific procedural knowledge into grounded skill packages (Figure 2, left), providing the common starting point for the maintenance experiments that follow.

Skill representation. A skill is a self-contained package that documents a reusable procedure in `SKILL.md`, with optional runnable `scripts` and supporting `references`. Repo2Skill adapts this format to repository settings by distilling repository-specific procedures into a skill set $\mathcal{S} = \{s_1, \ldots, s_m\}$. Provenance is recorded during distillation. The agent logs the source paths and symbols behind each claim as external metadata, which supports the grounding gate and later tracing of content affected by repository changes.

The distillation agent. The agent's ANALYZE stage performs static analysis to build a symbol index I_{sym} that maps repository symbols and file paths to source locations. This index anchors subsequent logic analysis and material collection. The distillation agent then runs a think–act–observe loop over a workspace state

$$w_t = (\mathcal{H}_t, A_t, L_t, M_t, \mathcal{S}_t),$$

where $\mathcal{H}_t$ is the interaction history and A_t, L_t, M_t, and $\mathcal{S}_t$ are the accumulated analysis report, logic read-map, material documents, and skill package. At each step, the agent chooses from

$$\mathcal{A}_{\text{distill}} = \{\text{ANALYZE}, \text{LOGIC}, \text{MATERIALS}, \text{DRAFT}, \text{VALIDATE}, \text{FINISH}\}.$$

ANALYZE runs first to construct I_{sym}, after which LOGIC derives a routing read-map and MATERIALS collects repository-grounded evidence over the same repository partitions. DRAFT jointly uses these intermediate products to author the skill package, VALIDATE applies the grounding gate, and FINISH terminates the run once validation succeeds. The stages follow this order by default, with earlier stages revisited only for recovery after a failure or validation rejection. Figure 2 illustrates this process through a simplified worked example.

Grounding gate. After DRAFT, VALIDATE applies structural and grounding checks to the candidate skill set by verifying the source paths and symbols referenced in the generated `SKILL.md` files and `scripts` against the V_1 repository. For candidates meeting the minimum reference count, validation fails once the fraction of unverifiable references exceeds a preset threshold, and only the candidates that pass proceed to expert selection and refinement. The gate therefore establishes source traceability, and its scope stops at reference resolution, since a referenced path or symbol can resolve even when the surrounding guidance uses it incorrectly. Semantic correctness and procedural coverage accordingly fall to the expert, who reviews both during selection and refinement.

Candidate generation and selection. For each transition, Claude-opus-4.6 [3] and GPT-5.4 [25] each produce one candidate V_1 skill set under the same staged Repo2Skill procedure, using only the pre-release repository. An expert selects the stronger candidate, then revises and verifies it against the V_1 repository. The resulting $\mathcal{S}_{V_1}$ is frozen before gold-label construction begins and is shared by all maintenance agents. Appendix B.2 details this process.

3.2 Maintenance Task

Once a repository advances from V_1 to V_2, parts of its V_1 skill set may no longer match the updated interfaces, defaults, file layouts, or procedures. Maintenance therefore requires mapping patch-level changes to the affected guidance and making targeted updates while preserving content that remains valid. Repo2Skill-Evo operationalizes this problem over official release transitions. The right panel of Figure 2 summarizes the maintenance setting and evaluation.

Task definition. Each maintenance instance is a tuple $(\mathcal{S}_{V_1}, \Delta_{V_1 \to V_2}, \mathcal{U}_{\text{maint}})$ formed by the fixed V_1 skill set $\mathcal{S}_{V_1}$, the official release patch $\Delta_{V_1 \to V_2}$, and the maintenance tool interface $\mathcal{U}_{\text{maint}}$ available to the agent. The agent produces an updated skill set $\mathcal{S}_{V_2}$ and is evaluated on the set-level transformation $\mathcal{S}_{V_1} \to \mathcal{S}_{V_2}$. Successful maintenance requires the three capabilities below.

- **Patch-to-skill localization.** Identify the skill content affected by patch-level changes to APIs, files, commands, configurations, or usage procedures.
- **Obsolete-knowledge removal.** Remove or revise stale references, signatures, examples, tables, and prose that are no longer valid under V_2.
- **Information retention.** Preserve skill content that remains valid, avoiding unsupported rewrites, dropped context, or wholesale regeneration that discards useful repository-specific knowledge.

These requirements motivate our patch-grounded removal metric and complementary final-state NL Judge.

3.3 Agent Scaffold

To attribute performance differences to model behavior, we evaluate every model in the same deliberately minimal single-agent environment. The scaffold provides six generic tools covering shell execution, bounded file discovery and inspection, string-based editing, and termination, without any task-specific localization or repair module. Each agent receives the same fixed V_1 skill set, official release patch, and standard operating procedure (SOP) prompt, and edits the workspace to produce $\mathcal{S}_{V_2}$. All runs use a pre-specified turn budget. The budget is stated at initialization, and the remaining budget is reported to the agent throughout execution. If the budget is exhausted before the agent finishes, the resulting partial skill set is still evaluated. This controlled scaffold enables comparison of localization, editing, and convergence behavior under the same external constraints.

3.4 Evaluation Metric

The headline evaluation is a strictly patch-grounded removal metric. It scores whether the agent removes or modifies the V_1 knowledge that the release patch directly shows to be obsolete, while penalizing edits to V_1 lines outside the patch-supported obsolete set. Since valid V_2 content may admit multiple realizations, additions and reformulations in the updated skill set are assessed by the complementary NL Judge described below.

Gold obsolete set. For each transition, we construct a **gold obsolete set** G_Δ containing the V_1 skill lines that the official $V_1 \to V_2$ release patch directly shows to be no longer valid under V_2. This includes content invalidated by changes to APIs, signatures, defaults, files, commands, configurations, schemas, behavioral contracts, or other repository behavior exposed by the patch. Obsolete content may appear in prose, examples, reference tables, or scripts. Throughout the metric, a line denotes one physical line of a skill file, and diff matches are made against the original V_1 lines without merging, splitting, or normalization. G_Δ is defined on the V_1 side of the transition: it identifies lines that require deletion or revision while leaving the form of a valid V_2 replacement unconstrained. We construct G_Δ by automated candidate discovery followed by line-level manual verification against direct patch evidence (Appendix B.3).

Removal precision, recall, and F_1. We evaluate the V_1-side removal target defined by G_Δ. Let R denote the set of V_1 skill lines that appear on the deletion side of the skill-set diff between $\mathcal{S}_{V_1}$ and the agent-produced $\mathcal{S}_{V_2}$. Thus, R captures both deletions and revisions to existing guidance, while newly added V_2 lines are not included. A gold obsolete line counts as hit when its original form appears on the deletion side of the $\mathcal{S}_{V_1} \to \mathcal{S}_{V_2}$ diff. We deliberately do not match newly generated "+" lines, since a valid post-release replacement may admit multiple semantically equivalent realizations, and final-state quality is instead assessed by the NL Judge. We compute

$$\text{Recall} = \frac{|G_\Delta \cap R|}{|G_\Delta|}, \qquad \text{Precision} = \frac{|G_\Delta \cap R|}{|R|}, \qquad F_1 = \frac{2|G_\Delta \cap R|}{|G_\Delta| + |R|}.$$

Recall measures the fraction of patch-verified stale lines that are removed or revised, whereas precision measures the fraction of changed V_1 lines that belong to G_Δ. Their harmonic mean balances stale-content coverage against edit restraint under the minimal-edit objective. All 105 transitions have $|G_\Delta| > 0$, so recall is always defined. When $R \neq \varnothing$ but $G_\Delta \cap R = \varnothing$, precision is zero, and when $R = \varnothing$, recall and F_1 are zero while precision is undefined.

For each transition and metric, avg@3 is the mean over three runs and best@3 is the maximum over the three runs, with undefined precision values omitted. We then macro-average these transition-level scores across the corpus. Recall and F_1 therefore include all 105 transitions, whereas macro precision excludes a transition only when precision is undefined in all three runs. Our headline score is avg@3 macro F_1, with macro recall and precision reported separately.

Complementary NL Judge. Because the patch-grounded metric scores the V_1-side removal target, the semantic correctness of replacement and newly added V_2 content falls to a complementary NL Judge [12, 20, 40]. For each run, the judge reads the release patch, the V_1 skills, the produced $\mathcal{S}_{V_2}$ skill set, and limited run metadata. The judge assigns five 0–2 scores. API checks whether the produced skills describe the correct V_2 API, FILES checks whether the skill prose, scripts, and references are mutually consistent, INFO checks whether still-valid V_1 guidance was retained, PROMPT checks adherence to the patch-grounded maintenance instructions, and NL checks whether the prose describes the post-release state directly. API correctness, file consistency, and retention are aggregated across impacted skills, while prompt adherence and NL correctness are package-level judgments. The Sum score (0–10) is the total over these five dimensions, with structurally invalid skill sets form-gated to zero.

4 Experiments

Our experiments center on whether agents can maintain repository skills across releases. We first conduct a focused utility study to assess whether the fixed V_1 skills encode repository-specific knowledge with meaningful downstream utility, thereby motivating their maintenance. We then evaluate maintenance performance and diagnose the sources of failure.

4.1 Setup

Maintenance data. Repo2Skill-Evo contains 105 selected official release transitions from 57 public GitHub repositories, with one fixed V_1 skill set per transition and 1,158 individual skills in total. Each transition pairs official V_1 and V_2 tags with the corresponding source diff. The corpus focuses on the fast-moving AI/ML and LLM-agent ecosystem, covering agent and RAG frameworks, training and inference stacks, and supporting infrastructure, while retaining several mature general-purpose libraries for breadth. We prioritize transitions with visible source-level changes that may affect repository-specific procedural knowledge (Appendix B.1). Across the corpus, the 105 gold obsolete sets contain 12,217 patch-verified stale V_1 skill lines, with individual sets ranging from 5 to 375 lines and a median of 92 (Appendix D.3).

Models and protocol. We evaluate Claude-opus-4.6, GLM-5.1 [7], GPT-5.4, Kimi-K2.5 [30], Doubao-Seed2-pro [29], and MiniMax-M2.5 [24]. Each model runs three times per transition under the same agent scaffold and 50-turn budget. For each transition, all runs receive the same fixed V_1 skill set and official $V_1 \to V_2$ release patch in a standardized tool environment.

Corpus composition. Figure 3 summarizes the corpus along two axes: repository domain and observed maintenance difficulty. By domain, the corpus includes 33 transitions from agent and RAG frameworks, 32 from training and inference infrastructure, 17 from data and classic ML libraries, 16 from compiler, kernel, and systems software, and 7 from general-purpose libraries and applications. For each transition, we average avg@3 F_1 across the six maintenance models and classify the resulting observed difficulty as EASY (≥ 0.65), MEDIUM ($[0.40, 0.65)$), or HARD (< 0.40). Appendix D.4 provides the classification criteria, per-transition assignments, per-model scores, and domain-level outcomes.

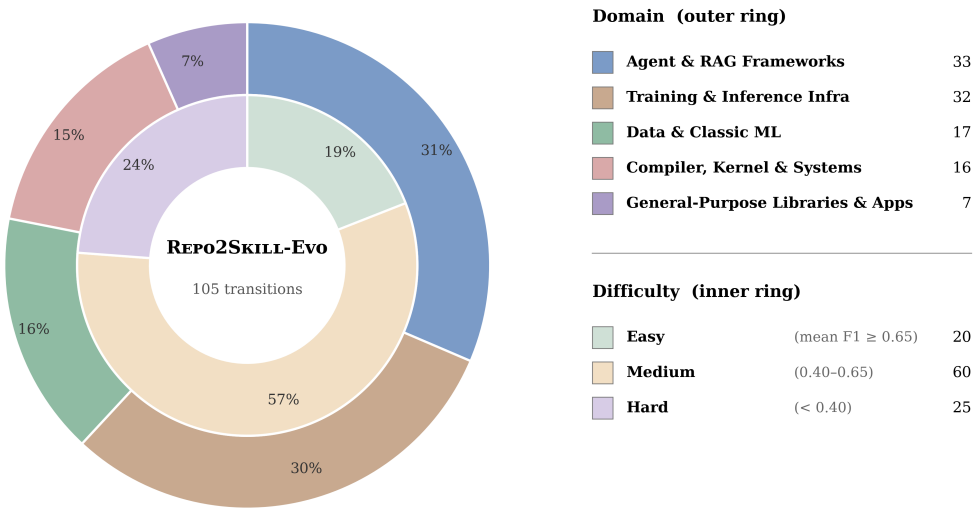

Figure 3 Composition of the Repo2Skill-Evo corpus, covering 105 release transitions across 57 repositories. The outer ring shows repository domain, and the inner ring shows observed maintenance difficulty, defined by the mean avg@3 F_1 across the six maintenance agents. Per-transition assignments and the full score matrix appear in Appendix D.4.

4.2 Motivating Study: Fixed Skills Provide Meaningful Repository Utility

Before evaluating whether repository skills can be maintained across releases, we first ask whether the fixed V_1 skills encode repository-specific knowledge worth maintaining. Prior work has shown that skill benefits vary substantially across tasks [8, 16]. We therefore evaluate the downstream utility of the Repo2Skill-Evo skills and examine where their gains are largest.

Experimental protocol. We randomly sample ten repositories from the corpus and define one artifact-generation task per repository, together with a repository-specific utility rubric that remains hidden from the model. We evaluate GPT-5.4 in a 2×2 ablation over source and skill access, with three runs in each of the four conditions `baseline`, `skill-only`, `source-only`, and `source+skill`. Appendix A defines these conditions along with the three contrasts and reports the per-repository scores. We use baseline utility as a coarse proxy for prior repository knowledge and split the repositories at the median into low- and high-baseline groups (Table 1).

Table 1 Repository-skill utility by baseline-utility group. Values are group means of artifact utility scores (0–10), manually scored under repository-specific rubrics, with three runs per repository and condition. Groups are formed by a median split of baseline utility. Skill and source deltas are relative to baseline; skill Δ_{src} is relative to source-only. Appendix A provides per-repository scores and cost statistics.

Group	#repos	baseline	skill-only	source-only	source+skill	skill Δ	source Δ	skill Δ_{src}
All repositories	10	5.88	8.68	8.64	9.01	+2.80	+2.76	+0.37
Low-baseline repositories	5	4.48	8.46	8.46	8.86	**+3.98**	+3.98	+0.40
High-baseline repositories	5	7.28	8.89	8.82	9.15	+1.61	+1.54	+0.33

Skills provide substantial downstream utility. Across all ten repositories, skill-only raises mean utility from 5.88 to 8.68, a gain of 2.80 points, and nearly matches source-only at 8.64. The fixed skills therefore provide task-relevant repository knowledge that the model does not consistently recover without external context.

Gains are largest at low baseline utility. Skill-only raises mean utility in the low-baseline group from 4.48 to 8.46, a gain of 3.98 points, compared with 1.61 points in the high-baseline group. Within this sample, skills provide the largest absolute gains on repositories where the model performs worst without source or skill access.

Skills provide comparable utility at substantially lower cost. Skill-only and source-only achieve nearly identical

mean utility (8.68 versus 8.64), but skill-only uses 51,821 mean tokens and 3.9 iterations, compared with 272,620 tokens and 10.8 iterations for source-only. When source is already available, adding skills increases mean utility from 8.64 to 9.01 while modestly reducing mean token usage (Appendix A). Together, these results show that the fixed V_1 skills encode useful, compact repository knowledge, motivating their maintenance across releases.

4.3 Agents Do Not Reliably Maintain Repository Skills across Releases

We now evaluate the maintenance task defined in Section 3.2. Table 2 reports the headline patch-grounded removal scores. The metric rewards the removal or revision of obsolete V_1 lines while penalizing edits to V_1 lines outside G_Δ.

Maintenance remains far from solved. Claude-opus-4.6 achieves the highest avg@3 macro F_1 at 69.7%, followed by GLM-5.1 at 64.3%, while the remaining four models fall below 60%. GPT-5.4 obtains the highest avg@3 macro recall at 74.6%, but its substantially lower macro precision of 55.7% limits its avg@3 macro F_1 to 58.8%. Kimi-K2.5, Doubao-Seed2-pro, and MiniMax-M2.5 achieve avg@3 macro F_1 scores of only 47.0%, 39.8%, and 29.9%, respectively.

These score profiles have direct maintenance consequences. Low recall leaves patch-verified stale guidance in the produced skill set, whereas low precision reflects broader editing outside the patch-grounded obsolete set and therefore weaker adherence to the task's minimal-edit objective. Repository-level macro averaging preserves the model ordering, and 95% paired repository-clustered bootstrap intervals for the three adjacent avg@3 F_1 differences among the top four models all exclude zero (Appendix D.2).

Table 2 Patch-grounded maintenance scores (%) over 105 release transitions and three runs per model. R, P, and F_1 denote transition-macro recall, precision, and F_1. avg@3 averages the three runs for each transition, whereas best@3 takes the per-transition maximum over the three runs before macro-averaging. Macro F_1 is averaged directly over transitions and is therefore not the harmonic mean of the displayed macro R and P. Models are sorted by avg@3 macro F_1; the best result in each column is shown in bold.

	avg@3 macro			best@3 macro		
Model	R	P	F_1	R	P	F_1
Claude-opus-4.6	70.4	**75.7**	**69.7**	79.3	82.2	**76.1**
GLM-5.1	64.1	72.4	64.3	78.1	80.9	73.9
GPT-5.4	**74.6**	55.7	58.8	**87.9**	67.9	70.4
Kimi-K2.5	40.4	71.6	47.0	55.0	84.9	60.7
Doubao-Seed2-pro	31.9	73.3	39.8	44.7	85.0	53.4
MiniMax-M2.5	22.6	74.9	29.9	36.6	**86.8**	46.0

Models occupy different points on the recall--precision tradeoff. GPT-5.4 removes or revises a broader set of V_1 lines and achieves the highest recall, but many of these changes fall outside G_Δ. Claude-opus-4.6 and GLM-5.1 maintain more balanced recall–precision profiles. The lower-scoring models tend toward high precision and low recall: MiniMax-M2.5, for example, reaches 74.9% precision but removes or revises only 22.6% of the verified stale content. High precision in this regime does not imply successful maintenance; it coexists with substantial under-editing. Section 4.4 connects these outcome profiles to affected-file coverage and edit-selection diagnostics derived from the execution traces and output diffs.

Repeated attempts reveal capability but not reliability. Selecting the highest F_1 among three runs for each transition raises macro F_1 by 6.4 points for Claude-opus-4.6 and by 9.6–16.1 points for the remaining models. These gains indicate that models sometimes produce substantially better updates than their mean single-run performance suggests. However, even best@3 reaches only 76.1% for the strongest model, and no other model exceeds 73.9%. Moreover, best@3 selects the strongest of three observed outcomes and therefore reflects attainable performance across repeated attempts rather than single-attempt reliability. Together with the within-transition run-to-run variability reported in Appendix D.2, the generally larger best@3–avg@3 gaps for the lower-scoring models indicate that maintenance quality is unstable across repeated attempts.

Complementary evaluation identifies the same upper tier. Table 3 reports five-dimensional NL Judge scores assigned by GPT-5.4 and Claude-opus-4.6. Both judges rank Claude-opus-4.6 first, GPT-5.4 second, and GLM-5.1 third. They therefore identify the same top-three set as the patch-grounded evaluation. All models produce at least some structurally invalid skill sets, with 7–26 invalid runs out of 315 per model. The NL Judge complements the headline metric by assessing the final skill set across five rubric dimensions, whereas the patch-grounded removal metric provides a direct and auditable measure of stale-content removal and off-target editing. Full dimension-level results appear in Appendix D.1.

Table 3 Complementary NL Judge scores over 315 runs per maintenance model. Each judge reports the mean 0–10 Sum score, with structurally invalid outputs form-gated to zero before averaging. Invalid reports the number of such outputs. The best score in each judge column is shown in bold.

Model	GPT-5.4 judge	Claude-opus-4.6 judge	Invalid
Claude-opus-4.6	**7.54**	**7.86**	9
GLM-5.1	7.09	6.98	7
GPT-5.4	7.18	7.31	26
Kimi-K2.5	6.49	6.24	14
Doubao-Seed2-pro	6.54	6.22	15
MiniMax-M2.5	6.01	5.27	8

The maintenance shortfall extends across the corpus. Transition difficulty varies substantially, but poor performance is not confined to a few isolated cases. Under the observed-difficulty stratification defined in Section 4.1, 25 of the 105 transitions fall below 0.40, 60 fall in $[0.40, 0.65)$, and only 20 reach at least 0.65 when averaging the six models' avg@3 F_1 scores. Thus, 85 transitions fall below the EASY threshold. Appendix D.4 provides the full stratification and representative high- and low-scoring transitions.

4.4 Failure Diagnosis: Coverage and Editing Bottlenecks

Maintenance can fail because agents miss affected skill files, edit too little or too broadly, or run out of turns before finishing. We analyze all 1,890 runs using execution traces and maintained-output diffs. Table 4 summarizes the resulting model-level diagnostics. Affected-file coverage measures the fraction of gold-affected skill files localized during a run. Removal load, $|R|/|G_\Delta|$, measures the volume of changed V_1 lines relative to the gold removal target, while off-target editing measures the fraction of changed V_1 lines outside G_Δ. Together, these diagnostics distinguish missed localization and under-editing from broad, poorly targeted editing. They are used only for post hoc analysis. Appendix C.1 provides the full definitions.

Table 4 Post hoc diagnostics over 315 runs per model. Coverage and removal load are mean ratios; full coverage, off-target editing, no-removal runs, and budget exhaustion are percentages; turns is the mean number of turns per run.

Model	mean file cov.	full file cov.	removal load	off-target	no-removal	budget exh.	turns
Claude-opus-4.6	0.95	75.9	1.01	24.0	0.6	1.9	25.3
GLM-5.1	0.95	78.1	1.01	27.6	0.6	41.3	42.4
GPT-5.4	0.93	69.2	2.84	44.3	0.0	0.0	17.8
Kimi-K2.5	0.74	25.7	0.60	28.6	2.9	47.3	43.5
Doubao-Seed2-pro	0.53	10.5	0.42	25.9	4.1	14.6	33.1
MiniMax-M2.5	0.51	9.5	0.30	22.7	20.0	68.6	47.0

Coverage is strongly associated with maintenance quality. Across all 1,890 runs, affected-file coverage correlates with both F_1 ($r = 0.650$) and recall ($r = 0.701$), and these associations persist after removing additive model and transition effects (Appendix C.1). Runs with full affected-file coverage average 67.3% F_1 and 72.1% recall, compared with 41.0% F_1 for partial-coverage runs. These results consistently associate incomplete localization with missed stale content.

Oracle localization improves performance but leaves substantial residual errors. We test localization more directly through an oracle ablation on the hardest-20 subset, restricted to transitions with at least 20 gold

obsolete lines. We append only the gold-affected skill roots and file paths to the original prompt, without providing patch hunks, line numbers, obsolete-line labels, or edit instructions.

Table 5 Oracle skill-file localization on the hardest-20 subset. Values are avg@3 macro scores (%). Gain columns report mean per-transition oracle-minus-baseline differences.

	Baseline			Oracle			Gain		
Model	R	P	F_1	R	P	F_1	ΔR	ΔP	ΔF_1
Claude-opus-4.6	55.4	64.4	53.6	63.5	68.2	62.3	+8.1	+3.8	+8.7
GLM-5.1	46.2	65.2	48.7	54.8	66.0	55.9	+8.6	+0.7	+7.2
GPT-5.4	65.6	47.9	48.2	73.3	50.5	54.6	+7.7	+2.6	+6.3
Kimi-K2.5	14.2	50.4	19.0	31.8	61.9	36.6	+17.6	+11.5	+17.6
Doubao-Seed2-pro	11.5	58.6	16.9	30.2	68.9	37.6	+18.7	+10.3	+20.8
MiniMax-M2.5	6.7	50.0	10.1	18.0	69.4	23.0	+11.2	+19.1	+12.9
Mean	33.3	56.1	32.8	45.3	64.1	45.0	+12.0	+8.0	+12.2

Averaged across the six models, avg@3 macro F_1 increases from 32.8% to 45.0%, a mean transition-level gain of 12.2 points (95% paired bootstrap CI: $[+6.8, +17.8]$), while recall increases by 12.0 points. Recall improves for every model. Yet even with oracle paths, the best model reaches only 62.3% F_1 on this subset. File-level localization is therefore important but insufficient: agents must still identify the affected claims within each file and select the appropriate edits. Appendix C.2 provides the full experimental details.

Models exhibit distinct failure profiles. Figure 4 visualizes the transition-level recall–precision tradeoff behind the model averages.

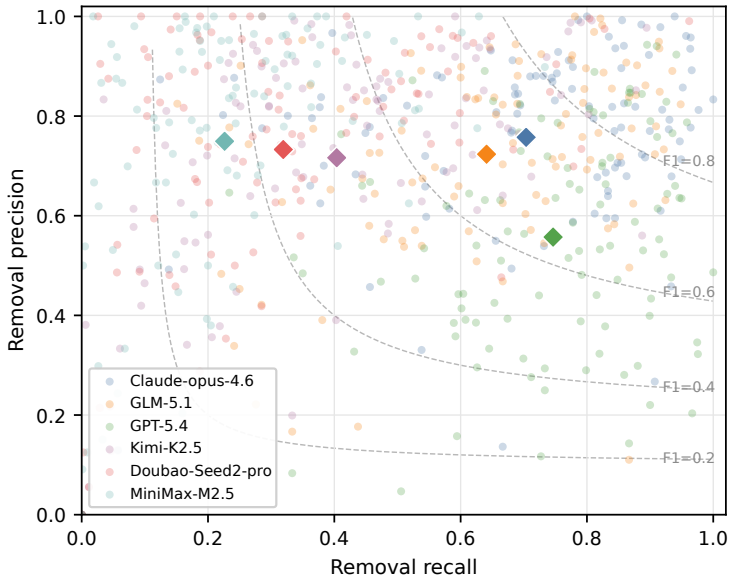

Figure 4 Removal recall versus precision for each model–transition pair after avg@3 aggregation. Pairs with undefined precision are omitted. Diamonds mark per-model means, and light curves show iso-F_1 contours.

The diagnostics reveal different routes to failure. GPT-5.4 achieves high affected-file coverage but has a removal load of 2.84 and the highest off-target editing rate, consistent with its high-recall, low-precision profile. In contrast, Kimi-K2.5, Doubao-Seed2-pro, and MiniMax-M2.5 show substantially lower coverage and removal loads, indicating that many failures arise before enough affected content is reached and edited. Kimi-K2.5 and MiniMax-M2.5 also frequently exhaust the turn budget. GLM-5.1 achieves high coverage and a near-unit removal load but exhausts the budget in 41.3% of runs, showing that finding the affected files does not by itself ensure successful completion. Claude-opus-4.6 exhibits the most balanced profile, combining high coverage, a near-unit removal load, and infrequent budget exhaustion. Overall, the results point to two

complementary maintenance bottlenecks: localizing the affected skill content and selecting edits that are sufficiently complete without becoming overly broad.

5 Limitations

Our dataset is a curated staleness-positive challenge set, so our results characterize maintenance difficulty on releases known to affect skills, with the prevalence of staleness across releases forming a separate question (Appendix B.1). Our evaluation measures patch-grounded maintenance fidelity, leaving the downstream utility of maintained skills to future work. Although the NL Judge provides a complementary assessment of final-state quality, we do not rerun the maintained skill sets on downstream repository tasks.

6 Conclusion

Repository-grounded skills turn repository-specific procedures into reusable external knowledge, but their validity does not persist automatically. As repositories evolve, previously correct guidance can become misleading while remaining loadable and retrievable. We call this release-driven failure mode *silent staleness*.

Repo2Skill-Evo operationalizes this lifecycle problem as a measurable maintenance task by pairing fixed V_1 skill sets with official release patches and evaluating the resulting updates against patch-grounded targets. Across 57 repositories and 105 selected release transitions, even the strongest of the six evaluated agents reaches only 69.7% avg@3 macro F_1. Trace and output analyses characterize two complementary maintenance bottlenecks: incomplete localization of affected skill files and imperfect edit selection. Oracle skill-file localization improves performance but leaves substantial residual errors, indicating that localization alone is insufficient. Together, these findings suggest that repository skills should be treated as versioned knowledge assets that carry the release they were distilled from and require maintenance as the repository evolves.

7 Contributions

Project Lead($\alpha - \beta$ order)

Chenyuan Duan (duanchenyuan26@stu.pku.edu.cn)

Ge Shi(23121732@bjtu.edu.cn)

Core Contributors($\alpha - \beta$ order)

Zineng Mao, Ge Zhang

Contributors($\alpha - \beta$ order)

Hao Liang, Yinzhu Piao, Yuchen Wu, Zhixin Yao

Corresponding($\alpha - \beta$ order)

Kaiyu Huang, Wenhao Huang, Linzhuang Sun

Shen Yan(sheny@bytedance.com)

Wentao Zhang(wentao.zhang@pku.edu.cn)

References

[1] Agent Skills. Agent Skills overview, 2026. URL https://agentskills.io/home. [Accessed 04-07-2026].

[2] Hussein Alrubaye, Mohamed Wiem Mkaouer, and Ali Ouni. Migrationminer: An automated detection tool of third-party java library migration at the method level. In *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*, pages 414–417. IEEE, 2019. doi: 10.1109/ICSME.2019.00072. URL https://doi.org/10.1109/ICSME.2019.00072.

[3] Anthropic. System card: Claude Opus 4.6, 2026. URL https://www.anthropic.com/claude-opus-4-6-system-card. [Accessed 04-07-2026].

[4] Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. SWE-CI: evaluating agent capabilities in maintaining codebases via continuous integration. *CoRR*, abs/2603.03823, 2026. doi: 10.48550/ARXIV.2603.03823. URL https://doi.org/10.48550/arXiv.2603.03823.

[5] Linfeng Fan, Yuan Tian, Ziwei Li, and Zhiwu Lu. Skill drift is contract violation: Proactive maintenance for LLM agent skill libraries. *CoRR*, abs/2605.10990, 2026. doi: 10.48550/ARXIV.2605.10990. URL https://doi.org/10.48550/arXiv.2605.10990.

[6] Haoyu Gao, Jai Lal Lulla, Hong Yi Lin, Sebastian Baltes, Christoph Treude, and Mansooreh Zahedi. From registry to repository: How ai agent skills are written, adapted, and maintained, 2026. URL https://arxiv.org/abs/2607.00911.

[7] GLM-Team. GLM-5: from vibe coding to agentic engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

[8] Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. Swe-skills-bench: Do agent skills actually help in real-world software engineering? *CoRR*, abs/2603.15401, 2026. doi: 10.48550/ARXIV.2603.15401. URL https://doi.org/10.48550/arXiv.2603.15401.

[9] Mohayeminul Islam, Ajay Kumar Jha, Ildar Akhmetov, and Sarah Nadi. Characterizing python library migrations. *Proc. ACM Softw. Eng.*, 1(FSE):92–114, 2024. doi: 10.1145/3643731. URL https://doi.org/10.1145/3643731.

[10] Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agentic skills - beyond tool use in LLM agents. *CoRR*, abs/2602.20867, 2026. doi: 10.48550/ARXIV.2602.20867. URL https://doi.org/10.48550/arXiv.2602.20867.

[11] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[12] Seungone Kim, Jamin Shin, Yejin Choi, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=8euJaTveKw.

[13] Raula Gaikovina Kula, Ali Ouni, Daniel M. Germán, and Katsuro Inoue. An empirical study on the impact of refactoring activities on evolving client-used apis. *Inf. Softw. Technol.*, 93:186–199, 2018. doi: 10.1016/J.INFSOF.2017.09.007. URL https://doi.org/10.1016/j.infsof.2017.09.007.

[14] Man Ho Lam, Chaozheng Wang, Hange Liu, Jingyu Xiao, Haau-sing Li, Jen-tse Huang, Terry Yue Zhuo, and Michael R. Lyu. Swe-chain: Benchmarking coding agents on chained release-level package upgrades. *CoRR*, abs/2605.14415, 2026. doi: 10.48550/ARXIV.2605.14415. URL https://doi.org/10.48550/arXiv.2605.14415.

[15] Hanyu Li, Yichi Zhang, Speed Zhu, Hang Su, Jun Zhu, and Yinpeng Dong. Repomirage: Probing repository context reasoning in code agents with perturbations. *CoRR*, abs/2605.26177, 2026. doi: 10.48550/ARXIV.2605.26177. URL https://doi.org/10.48550/arXiv.2605.26177.

[16] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianfu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-Chung Lee. Skillsbench: Benchmarking how

well agent skills work across diverse tasks. *CoRR*, abs/2602.12670, 2026. doi: 10.48550/ARXIV.2602.12670. URL https://doi.org/10.48550/arXiv.2602.12670.

[17] Zhiwei Li and Yong Hu. Skillhone: A harness for continual agent skill evolution through persistent decision history. *CoRR*, abs/2606.08671, 2026. doi: 10.48550/ARXIV.2606.08671. URL https://doi.org/10.48550/arXiv.2606.08671.

[18] Tianyang Liu, Canwen Xu, and Julian J. McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=pPjZIOuQuF.

[19] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

[20] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using gpt-4 with better human alignment. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 2511–2522. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.153. URL https://doi.org/10.18653/v1/2023.emnlp-main.153.

[21] Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, Xiyu Ren, Ruan Chenyu, Haoran Li, and Yangqiu Song. Skillrevise: Improving LLM-authored agent skills via trace-conditioned skill revision. *CoRR*, abs/2606.01139, 2026. doi: 10.48550/ARXIV.2606.01139. URL https://doi.org/10.48550/arXiv.2606.01139.

[22] Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: in-context agentic reinforcement learning for skill internalization. *CoRR*, abs/2604.02268, 2026. doi: 10.48550/ARXIV.2604.02268. URL https://doi.org/10.48550/arXiv.2604.02268.

[23] Zijian Lu, Yiping Zuo, Yupeng Nie, Xin He, Weibei Fan, Lianyong Qi, and Shi Jin. Contractskill: Repairable contract-based skills for multimodal web agents. *CoRR*, abs/2603.20340, 2026. doi: 10.48550/ARXIV.2603.20340. URL https://doi.org/10.48550/arXiv.2603.20340.

[24] MiniMax. MiniMax-M2.5 model card, 2026. URL https://huggingface.co/MiniMaxAI/MiniMax-M2.5. [Accessed 04-07-2026].

[25] OpenAI. GPT-5.4 thinking system card, 2026. URL https://openai.com/index/gpt-5-4-thinking-system-card/. [Accessed 04-07-2026].

[26] Hongji Pu, Xinyuan Song, and Liang Zhao. Skillops: Managing LLM agent skill libraries as self-maintaining software ecosystems. *CoRR*, abs/2605.13716, 2026. doi: 10.48550/ARXIV.2605.13716. URL https://doi.org/10.48550/arXiv.2605.13716.

[27] PyTorch. Repository agent skills in the PyTorch codebase, 2026. URL https://github.com/pytorch/pytorch/tree/main/.claude/skills. Public skill directory maintained under the repository's `.claude/skills/` path. [Accessed 04-07-2026].

[28] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

[29] Bytedance Seed. Seed2.0 model card: Towards intelligence frontier for real-world complexity. *CoRR*, abs/2607.00248, 2026. doi: 10.48550/ARXIV.2607.00248. URL https://doi.org/10.48550/arXiv.2607.00248.

[30] Kimi Team. Kimi K2.5: visual agentic intelligence. *CoRR*, abs/2602.02276, 2026. doi: 10.48550/ARXIV.2602.02276. URL https://doi.org/10.48550/arXiv.2602.02276.

[31] Cédric Teyton, Jean-Rémy Falleri, Marc Palyart, and Xavier Blanc. A study of library migrations in java. *J. Softw. Evol. Process.*, 26(11):1030–1052, 2014. doi: 10.1002/SMR.1660. URL https://doi.org/10.1002/smr.1660.

[32] Minh V. T. Thai, Tue Le, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. SWE-EVO: benchmarking coding agents in long-horizon software evolution scenarios. *CoRR*, abs/2512.18470, 2025. doi: 10.48550/ARXIV.2512.18470. URL https://doi.org/10.48550/arXiv.2512.18470.

[33] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. A large-scale empirical study on code-comment inconsistencies. In Yann-Gaël Guéhéneuc, Foutse Khomh, and Federica Sarro, editors, *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 53–64. IEEE / ACM, 2019. doi: 10.1109/ICPC.2019.00019. URL https://doi.org/10.1109/ICPC.2019.00019.

[34] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE):801–824, 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

[35] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

[36] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[37] Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *CoRR*, abs/2604.01687, 2026. doi: 10.48550/ARXIV.2604.01687. URL https://doi.org/10.48550/arXiv.2604.01687.

[38] 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. *CoRR*, abs/2605.19576, 2026. doi: 10.48550/ARXIV.2605.19576. URL https://doi.org/10.48550/arXiv.2605.19576.

[39] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In Maria Christakis and Michael Pradel, editors, *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, pages 1592–1604. ACM, 2024. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

[40] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

[41] Yifan Zhou, Zhentao Zhang, Ziming Cheng, Shuo Zhang, Qizhen Lan, Zhangquan Chen, Zhi Yang, Qianyu Xu, Ronghao Chen, Huacan Wang, and Sen Hu. Skillgenbench: Benchmarking skill generation pipelines for LLM agents. *CoRR*, abs/2605.18693, 2026. doi: 10.48550/ARXIV.2605.18693. URL https://doi.org/10.48550/arXiv.2605.18693.

[42] Yingli Zhou, Wang Shu, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. A comprehensive survey on agent skills: Taxonomy, techniques, and applications. *CoRR*, abs/2605.07358, 2026. doi: 10.48550/ARXIV.2605.07358. URL https://doi.org/10.48550/arXiv.2605.07358.

Appendix

A Skill-Gain Utility Study

Section 4.2 reports the aggregate findings of the motivating utility study. Here we provide the experimental protocol, per-repository results, baseline-utility grouping procedure, and execution costs.

Experimental protocol. Each of the ten sampled repositories is represented once. GPT-5.4 performs three runs in each cell of a 2×2 ablation over access to the fixed V_1 skill set and the V_1 repository checkout:

- **baseline**: neither source nor skill is available;
- **skill-only**: the skill set is available and the source is unavailable;
- **source-only**: the source is available and the skill set is unavailable;
- **source+skill**: both the source and the skill set are available.

The four conditions use the same repository-specific task, evaluation rubric, agent scaffold, and execution budget, differing only in source and skill access. Skill Δ is `skill-only` minus `baseline`, source Δ is `source-only` minus `baseline`, and skill Δ_{src} is `source+skill` minus `source-only`. These contrasts measure, respectively, the utility of skills without source access, the utility of source access without skills, and the incremental utility of skills when source is already available. Table 1 reports grouped means over the same per-repository scores shown in Table 6.

Table 6 Per-repository artifact utility scores for the ten-repository motivating study. Scores are 0–10 means over three runs. Skill Δ is `skill-only` minus `baseline`, source Δ is `source-only` minus `baseline`, and skill Δ_{src} is `source+skill` minus `source-only`. Rows are ordered by baseline utility, with the median split separating the low- and high-baseline groups.

Repository	baseline	skill-only	source-only	source+skill	skill Δ	source Δ	skill Δ_{src}
`agentmemory`	3.43	8.57	8.83	9.07	+5.13	+5.40	+0.23
`oh-my-pi`	4.33	8.73	7.90	9.10	+4.40	+3.57	+1.20
`harbor`	4.70	8.20	8.50	8.63	+3.50	+3.80	+0.13
`deer-flow`	4.73	8.47	9.07	9.17	+3.73	+4.33	+0.10
`hermes-agent`	5.20	8.33	8.00	8.33	+3.13	+2.80	+0.33
`browser-use`	6.37	8.70	8.73	9.00	+2.33	+2.37	+0.27
`lightrag`	6.83	8.67	9.13	9.17	+1.83	+2.30	+0.03
`mem0`	7.17	8.60	8.67	9.00	+1.43	+1.50	+0.33
`langgraph`	7.83	9.17	8.70	9.20	+1.33	+0.87	+0.50
`pydantic`	8.20	9.33	8.87	9.40	+1.13	+0.67	+0.53
Mean	5.88	8.68	8.64	9.01	+2.80	+2.76	+0.37

Baseline-utility grouping. We use baseline utility as a coarse operational proxy for repository-specific prior knowledge and split the repositories at the median. The five low-baseline repositories are `agentmemory`, `oh-my-pi`, `harbor`, `deer-flow`, and `hermes-agent`. The remaining five form the high-baseline group. Group means appear in Table 1, and the per-repository values from which they are computed appear in Table 6. Because baseline utility may also reflect task difficulty, and because lower-baseline tasks have more headroom on the bounded 0–10 scale, we interpret this grouping descriptively.

Execution cost. Table 7 reports utility and execution cost for all four conditions. Each row aggregates 30 runs: three runs on each of ten repositories.

Table 7 quantifies the cost reduction noted in Section 4.2. Adding skills on top of source access reduces mean tokens from 272,620 to 237,020 and mean iterations from 10.8 to 9.3, so the +0.37-point utility gain of `source+skill` over `source-only` comes at slightly lower cost.

Table 7 Artifact utility and execution cost across the four access conditions. Values are means over ten repositories and three runs per repository and condition.

Condition	Utility	Mean tokens	Mean iterations
Baseline	5.88	21,057	4.4
Skill-only	8.68	51,821	3.9
Source-only	8.64	272,620	10.8
Source+skill	9.01	237,020	9.3

B Maintenance Dataset Construction

Repo2Skill-Evo is constructed before any maintenance model is evaluated. The construction proceeds in four steps:

1. **Select release transitions.** We identify official releases with publicly available source diffs (Appendix B.1).
2. **Materialize release instances.** Each transition records its repository, V_1 and V_2 version tags, commit hashes, and official $V_1 \to V_2$ release patch.
3. **Construct the fixed V_1 skill set.** Repo2Skill produces candidate skill sets from the pre-release repository, after which an expert selects, revises, and verifies the final $\mathcal{S}_{V_1}$ used by all maintenance agents (Appendix B.2).
4. **Construct the gold obsolete set.** We identify the V_1 skill lines that the release patch directly shows to be invalid under V_2, producing the patch-grounded gold set G_Δ (Appendix B.3). Transitions with $G_\Delta = \varnothing$ are excluded, yielding the final staleness-positive corpus.

Gold annotations are stored separately from the maintenance instances and are used only for evaluation. Maintenance agents receive neither G_Δ nor its annotation-side representation.

B.1 Transition Selection and Construction

Source and scope. Beyond the corpus-level counts given in Section 4.1, each transition records the repository, the V_1 and V_2 version tags, the base and target commit hashes, and the corresponding public source diff, making every instance traceable to an upstream release. Selection favors actively maintained projects; among their releases we prefer diffs that touch interfaces, defaults, file layouts, commands, configurations, or documented repository behavior, since these are the change classes that can invalidate procedural guidance. The complete repository and release list is included in the released materials.

Recency and contamination. The selected transitions span a broad range of version histories, including releases that postdate the knowledge cutoffs of some evaluated models. We do not assume that the repositories themselves were absent from model pretraining. However, the maintenance task turns on aligning a fixed V_1 skill set with the changes in one specific $V_1 \to V_2$ release patch, so performance depends on the evidence that the patch itself supplies. We release version tags and commit hashes to support independent dating and auditing.

B.2 Fixed V_1 Skill Set

Section 3 states the candidate-generation and expert-selection procedure. Here we record the two properties that matter for the validity of the benchmark. First, *isolation*: both candidates are generated from the pre-release repository alone, and the expert revision draws on the same V_1 evidence, so the release patch and the gold obsolete labels remain outside the construction of $\mathcal{S}_{V_1}$ from end to end. Second, *review scope*: the expert reviews the package as a whole, including `SKILL.md`, scripts, and references. Cited paths and symbols are checked against the repository, procedural guidance is reviewed for factual consistency, and the package must satisfy the structural and grounding requirements before it is frozen.

B.3 Gold Obsolete Set Construction

Candidate discovery and manual verification. In the automated stage, a script extracts repository-specific interfaces referenced by the V_1 skill set, including symbols, paths, commands, and configuration keys, and matches them against changed lines in the release patch. This stage surfaces potentially affected skill content and is used only to assist candidate discovery. For every transition, the V_1 skill set is then inspected together with the corresponding release patch. Candidate lines whose documented guidance remains valid under V_2 are removed, while stale content missed by the automatic pass is added, including semantically expressed claims and lines whose validity depends on an affected interface or behavior. Every final line in G_Δ is manually verified against direct patch evidence before being used as a gold removal target.

Representative obsolete-content categories. Table 8 summarizes common sources of skill invalidation. These categories sample the range of changes observed across transitions, and membership in G_Δ follows from direct patch evidence for each individual line.

Table 8 Representative sources of V_1 skill obsolescence captured by G_Δ.

Repository change	Example effect on V_1 guidance
API or interface change	Referenced functions, classes, modules, or interfaces are removed or renamed
Signature or default change	Arguments, defaults, return contracts, or invocation patterns become invalid
File or entry-point change	Referenced paths, scripts, modules, or entry points move or disappear
Command or configuration change	Flags, commands, configuration keys, or schemas change
Behavioral contract change	Previously documented behavior is contradicted by the updated implementation

The resulting stale knowledge is not limited to the line that explicitly names the changed repository element. When a patch-supported change also invalidates dependent instructions, examples, table entries, or script statements, those lines are included in G_Δ as well.

Example. Consider the transition `flask-2.2.5-2.3.0`. A V_1 skill, `app-env-config`, describes environment mode using `FLASK_ENV`, the `ENV` configuration key, and the `app.env` property. The release removes this environment-mode surface, making the corresponding V_1 guidance obsolete. G_Δ therefore includes the affected reference-table rows, prose instructions that use `app.config["ENV"]`, and script lines that set `os.environ["FLASK_ENV"]` or access `app.env`. This example illustrates how a single release change can invalidate repository-specific guidance expressed across multiple files and content forms within a skill package.

C Evaluation Protocol and Diagnostics

C.1 Behavioral Diagnostics

Table 9 defines the post hoc diagnostics used in Section 4.4. Affected-file coverage is derived from agent traces and maintained outputs, while the remaining diff-based diagnostics are computed from the $\mathcal{S}_{V_1} \to \mathcal{S}_{V_2}$ diff. A gold-affected skill file is a file in $\mathcal{S}_{V_1}$ containing at least one line in G_Δ, and a file counts as localized when the agent directly reads it, explicitly names or targets it in the interaction trace, or modifies it in the maintained output. Localization therefore requires evidence that the agent attended to the file, which places paths appearing only in tool-returned search results outside the criterion. Affected-file coverage is the fraction of gold-affected skill files meeting this criterion.

Budget exhaustion covers runs that reach the 50-turn limit while still acting on the workspace, so a run whose final turn is a standalone `finish` counts as a completed run even on turn 50. Partial outputs from budget-exhausted runs enter the evaluation on the same terms as complete ones.

Coverage-association robustness. The pooled correlations reported in Section 4.4 use all 1,890 runs. As a robustness check, we first average the three runs for each model–transition pair, yielding 630 observations. We then residualize affected-file coverage, F_1, and recall with respect to additive model and transition fixed effects and compute correlations between the resulting residuals.

Table 9 Definitions of the post hoc diagnostics reported in Table 4.

Metric	Definition
`mean file cov.`	Run-level affected-file coverage, reported as the mean across runs.
`full file cov.`	Run-level indicator that every gold-affected skill file is localized, reported as the percentage of runs satisfying the indicator.
`removal load`	Run-level ratio $\lvert R\rvert/\lvert G_\Delta\rvert$, reported as the mean across runs.
`off-target`	Run-level fraction $\lvert R \setminus G_\Delta\rvert/\lvert R\rvert$, defined when $R \neq \varnothing$ and reported as the mean over defined runs.
`no-removal`	Run-level indicator that $R = \varnothing$, reported as the percentage of runs satisfying the indicator.
`budget exh.`	Run-level indicator that `run_outcome=max_iterations`, corresponding to 50 agent turns without a standalone `finish`, reported as the percentage of runs satisfying the indicator.
`turns`	Number of agent turns in a run, reported as the mean across runs.

Coverage remains positively associated with F_1 ($r = 0.477$) and recall ($r = 0.455$). The associations are attenuated relative to the pooled run-level correlations in Section 4.4, indicating that the relationship holds within models as well as across them.

Removal-load stratification. Runs with removal load in $[0.75, 1.25)$ average 77.3% F_1, compared with 25.0% for runs below 0.5. Runs with removal load at or above 1.25 achieve 80.8% recall but 44.7% precision. These patterns further distinguish insufficient editing from broad editing that improves stale-content coverage at the expense of edit selectivity.

C.2 Oracle Skill-File Localization Ablation

The oracle ablation in Section 4.4 tests the contribution of skill-file localization more directly. We first restrict the 105 transitions to those with at least 20 gold obsolete lines. Among the eligible transitions, we rank them by the mean baseline avg@3 F_1 across the six maintenance models and select in ascending order, allowing at most one transition per repository, until 20 transitions are retained. This produces the hardest-20 subset without allowing multiple releases from the same repository to dominate it.

For each selected transition and model, we keep the original maintenance scaffold, tools, 50-turn budget, release patch, and fixed V_1 skill set unchanged, so the oracle hint described in Section 4.4 is the only intervention. Agents must still inspect the identified files, trace the relevant release changes, and determine the required updates. Each of the six models is run three times per transition, yielding 360 oracle runs.

We compare the oracle runs with the original baseline runs restricted to the same 20 transitions using the identical patch-grounded removal metric. Per-model confidence intervals are bootstrapped over the 20 transition-level ΔF_1 values. For the overall interval, we first average ΔF_1 across the six models within each transition and bootstrap the resulting 20 transition-level means. Baseline and oracle precision are each aggregated over transitions for which precision is defined in the corresponding condition. The reported ΔP instead averages per-transition oracle-minus-baseline differences where both values are defined, and therefore need not equal the difference between the two displayed aggregate precision values.

C.3 Maintenance and Judge Instructions

We release the full instructions with the accompanying materials. Baseline maintenance agents receive the fixed V_1 skill set and official release patch alone, with gold-affected paths supplied only in the oracle localization ablation of Appendix C.2.

Maintenance instruction. The maintenance agent is instructed to read the existing skill set and official release patch, make minimal patch-grounded updates, cover affected prose, scripts, and references, and describe the post-release state directly rather than narrating the migration. It should preserve unaffected package structure unless the release requires a change, and it should finish within the fixed 50-turn budget. The total and remaining turn budget are reported throughout execution.

NL Judge instruction. The NL Judge works from the official release patch, the fixed V_1 skill set, the produced V_2 skill set, and limited run metadata alone, so its assessment stays independent of G_Δ and the patch-grounded removal scores. The full rubric is included in the released materials.

D Additional Results and Corpus Statistics

D.1 Full NL Judge Results

Table 10 reports the five dimension-level scores underlying the NL Judge results in Table 3. Form-gated outputs contribute zero to each dimension and to the Sum score. Values are averaged over all 315 runs for each maintenance model.

Table 10 Full NL Judge results. Sum is the 0–10 total over five 0–2 dimensions: API correctness, FILES consistency, INFO retention, PROMPT adherence, and NL current-state prose correctness. Invalid reports the number of form-gated runs.

	Dimension scores (0–2)					Total	Count
Model	API	Files	Info	Prompt	NL	Sum	Invalid
GPT-5.4 judge							
Claude-opus-4.6	1.38	1.00	1.75	1.65	1.76	**7.54**	9
GLM-5.1	1.20	1.03	1.76	1.37	1.74	7.09	7
GPT-5.4	1.25	1.04	1.52	1.66	1.70	7.18	26
Kimi-K2.5	1.06	0.92	1.74	1.20	1.57	6.49	14
Doubao-Seed2-pro	1.05	0.96	1.81	1.23	1.49	6.54	15
MiniMax-M2.5	0.93	1.02	1.86	0.92	1.28	6.01	8
Claude-opus-4.6 judge							
Claude-opus-4.6	1.38	1.24	1.60	1.84	1.81	**7.86**	9
GLM-5.1	1.20	1.09	1.61	1.36	1.72	6.98	7
GPT-5.4	1.24	1.26	1.32	1.76	1.73	7.31	26
Kimi-K2.5	0.99	0.89	1.53	1.24	1.59	6.24	14
Doubao-Seed2-pro	0.94	0.90	1.54	1.33	1.52	6.22	15
MiniMax-M2.5	0.86	0.79	1.53	0.86	1.23	5.27	8

D.2 Statistical Robustness of the Main Results

The headline metric gives equal weight to the 105 release transitions, but 28 of the 57 repositories contribute more than one transition. We therefore examine robustness to repository-level clustering, alternative repository-level aggregation, and run-to-run variability.

Repository-clustered confidence intervals. We resample the 57 repositories with replacement, retaining all transitions from each sampled repository, and recompute avg@3 macro F_1 over 10,000 bootstrap samples. Table 11 reports the resulting percentile 95% confidence intervals.

Repository-level macro average. As an alternative aggregation, we first average transition-level avg@3 F_1 within each repository and then average across the 57 repositories. This removes the additional weight given to repositories that contribute multiple transitions. The resulting repository-level macro scores preserve the model ordering and differ from the transition-level scores by only +0.9 to +3.4 points.

Paired repository-clustered bootstrap. We further assess the three adjacent differences among the top four models using the same repository-level resampling while pairing scores within each transition. All three 95% confidence intervals exclude zero: Claude-opus-4.6 exceeds GLM-5.1 by 5.4 points ($[+2.8, +8.0]$), GLM-5.1 exceeds GPT-5.4 by 5.4 points ($[+2.0, +8.7]$), and GPT-5.4 exceeds Kimi-K2.5 by 11.9 points ($[+8.1, +15.8]$).

Run-to-run variance. For each model and transition, we compute the standard deviation of F_1 across the three runs and then average over transitions. The resulting values are 6.1 for Claude-opus-4.6, 9.0 for GLM-5.1, 11.6 for GPT-5.4, 12.0 for Kimi-K2.5, 11.9 for Doubao-Seed2-pro, and 14.7 for MiniMax-M2.5. Run-to-run variability is generally larger for the lower-scoring models, providing a complementary view of maintenance reliability alongside avg@3 and best@3.

Table 11 Robustness of the headline avg@3 macro F_1 (%). "Transition-macro" is the main-text aggregation with equal weight per transition; the 95% CI is obtained by repository-clustered bootstrap. "Repository-macro" averages transition-level scores within each repository before averaging across repositories.

Model	Transition-macro	95% CI (cluster)	Repository-macro
Claude-opus-4.6	69.7	[65.7, 73.8]	72.7
GLM-5.1	64.3	[60.5, 68.2]	67.2
GPT-5.4	58.8	[55.1, 62.8]	62.3
Kimi-K2.5	47.0	[42.8, 51.3]	49.0
Doubao-Seed2-pro	39.8	[35.3, 44.8]	40.7
MiniMax-M2.5	29.9	[26.0, 34.0]	31.8

D.3 Obsolete-Set Size Distribution

Table 12 summarizes the size of the patch-verified obsolete set G_Δ across the 105 release transitions. The corpus contains 12,217 obsolete V_1 skill lines in total, ranging from 5 to 375 per transition, with a mean of 116.4 and quartiles of 46, 92, and 156. Only 7 transitions contain fewer than 20 obsolete lines, whereas 48 contain at least 100. Thus, most selected transitions require updating a nontrivial amount of skill content.

Table 12 Distribution of the patch-verified obsolete-set size $|G_\Delta|$ over the 105 release transitions.

$\lvert G_\Delta \rvert$ (obsolete V_1 skill lines)	Transitions	Share
5–19	7	6.7%
20–49	22	21.0%
50–99	28	26.7%
100–199	29	27.6%
≥ 200	19	18.1%
All	105	100%

D.4 Corpus Composition and Difficulty

Section 4.1 summarizes the corpus along repository-domain and observed difficulty axes. Here we describe the corresponding grouping criteria and provide representative transitions. The released materials include the full per-transition assignments and underlying model scores.

Difficulty stratification. For each transition, we compute avg@3 F_1 for each of the six maintenance models and average the six values to obtain an observed tractability score. We classify transitions as HARD (< 0.40), MEDIUM ($[0.40, 0.65)$), and EASY (≥ 0.65), yielding 25, 60, and 20 transitions, respectively. The thresholds fall in relatively sparse regions of the observed score distribution and are used only for descriptive analysis. Mean tractability scores are 29.9% for HARD, 53.0% for MEDIUM, and 74.5% for EASY. Neither release-patch size nor $|G_\Delta|$ enters the definition.

Table 13 lists representative transitions from the two ends of this observed difficulty spectrum.

Table 13 Representative hardest and easiest transitions ranked by mean avg@3 F_1 across the six maintenance models. Gold lines reports $|G_\Delta|$.

Group	Transition	Gold lines	Mean F_1
Hard	`DeepSpeed-0.17.6-0.18.0`	9	5.9
Hard	`peft-v0.15.0-v0.16.0`	29	11.5
Hard	`agentmemory-0.9.13-0.9.16`	5	12.1
Hard	`pydantic-v2.7.1-v2.9.1`	29	18.7
Hard	`datasets-3.0.0-4.0.0`	144	21.6
Easy	`openhuman-v0.53.35-v0.53.40`	35	88.9
Easy	`langchain-langchain-core==1.0.7-langchain-core==1.1.0`	97	85.3
Easy	`milvus-v2.5.10-v2.5.15`	72	83.1
Easy	`oh-my-pi-15.1.3-15.2.4`	60	79.9
Easy	`warp-v1.8.0-v1.8.1`	199	76.8

Domain composition. Each of the 57 repositories is assigned to one of five domains, and every transition inherits the domain of its repository. The corpus contains 19 agent and RAG framework repositories with 33 transitions, 15 training and inference infrastructure repositories with 32 transitions, 9 data and classic ML repositories with 17 transitions, 9 compiler, kernel, and systems repositories with 16 transitions, and 5 general-purpose library and application repositories with 7 transitions.

Domain-level outcomes. Mean F_1 varies across these domains: 53.8% for agent and RAG frameworks, 46.4% for training and inference infrastructure, 57.8% for data and classic ML, 51.6% for compiler, kernel, and systems software, and 49.5% for general-purpose libraries and applications. Training and inference infrastructure is disproportionately represented in the HARD stratum, contributing 9 of the 25 HARD transitions but only 2 of the 20 EASY transitions. These domain assignments are used only for descriptive corpus analysis and do not enter the construction of G_Δ, the maintenance runs, or any headline metric.