They Are Not Static: A Survey of Dynamic Agent Skills

Yubo Li
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
yubol@andrew.cmu.edu

Abstract

LLM agents increasingly externalize procedural knowledge into reusable *skills*: code, natural-language procedures, SKILL.md packages, workflow graphs, or learned adapters. This shifts adaptation from prompt or weight updates to library updates that change what future policies can retrieve, compose, execute, and trust. We survey a 94-paper 2023–2026 dynamic-skill audit set and argue that dynamic-skill systems are best understood as *lifecycle-managed, verified, evolving artifact stores*. The paper makes three contributions. *First*, we extend the options-style 4-tuple skill formalism to a 7-tuple $\langle C, \pi, T, R, \varphi, \nu, \prec \rangle$ and a library transition $\mathcal{L}_{t+1} = \mathrm{Apply}(u_t(\tau_t, r_t), \mathcal{L}_t)$ described by a ten-operator vocabulary. *Second*, we organize the corpus into eight lifecycle stages and eight system families, separating where each family invests engineering budget from what failure mode it leaves behind. *Third*, we synthesize five evidence-graded patterns—admission gates matter, verifier quality is load-bearing in skill-aware RL, flat retrieval can drop at moderate library scale, maintenance appears load-bearing after growth, and write-time abstraction often beats read-time alone—and translate them into a five-item reporting checklist for future dynamic-skill papers. The most consistent message across the literature is not that more skills help, but that what enters the library, under what gate, and with what abstraction is what determines whether a growing store becomes a reusable capability substrate or a polluted memory.

Keywords: LLM agents, agent skills, self-evolving agents, skill libraries, lifecycle management

1 Introduction

LLM agents are moving from chat-style assistance into operational workflows: they browse and manipulate web interfaces, write and repair software, operate desktop and computer-use environments, run multi-step research pipelines, and assist domain workflows in recommendation and medical imaging. Across these settings the bottleneck is not whether the model can reason about a single task, but whether the agent can reuse procedural knowledge across recurring tasks instead of re-deriving the same strategy in every context window.

Agent skills [7, 27, 60] answer this reuse problem with externally invocable procedural artifacts. A skill may be a function, a SKILL.md package, a workflow graph, or a learned adapter, but its role is the same: preserve a reusable way of acting so that a future agent can retrieve, compose, and execute it. By 2026, the ecosystem includes function-calling schemas, MCP/plugin manifests, and registries with 10^4–10^5 artifacts [51, 77, 95].

The static-library failure mode. Many libraries are still treated as *static*: authored once, versioned rarely, and disconnected from the trajectories that reveal whether a skill remains correct, useful, or safe. Static libraries pay authoring cost per skill before knowing which skills will matter; freeze an implicit verifier at authoring time; concede a moderate-library-size retrieval drop; lose the trajectory provenance that justified admission; and assume the deployment distribution will not drift. Recent work [3, 34, 48, 49, 53, 62, 74, 92] rejects all five of these assumptions by treating the library itself as a learning object.

Thesis. Dynamic-skill systems are *lifecycle-managed, verified, evolving artifact stores*. Skills are created from evidence, proposed as edits, verified, admitted, organized, retrieved or composed, maintained, sometimes distilled, and governed through provenance and rollback. Papers differ mainly in which lifecycle stages they implement, which signal drives edits, and where they place the verifier.

Relation to adjacent surveys. Three prior surveys cover overlapping territory and we do not duplicate their scope. SoK-Skills [27] systematizes agent-skill terminology and the static 4-tuple options view; we adopt and extend that 4-tuple with $(\varphi, \nu, \prec)$ to make library-level dynamics expressible. Xu and Yan [77] provide a broad position paper on agent skills, applications, and security; we narrow the target to lifecycle-managed artifact stores. Fang et al. [16] treat memory, prompt, weight, and tool updates uniformly under a single self-evolution umbrella; we restrict scope to externally invocable skill artifacts and analyze their lifecycle in detail. Zheng et al. [94] is memory-centric and emphasizes accumulation across episodes; we focus on the gating, maintenance, and provenance of artifacts that are meant to be *invoked* rather than retrieved as context. Relative to these, the contributions

specific to this paper are the operator vocabulary, lifecycle architecture, evidence-graded patterns, and reporting checklist; the master coding sheet (Appendix D) gives a per-system audit trail adjacent surveys do not provide.

Contributions and scope. This paper makes three contributions, drawing on a 94-paper 2023–2026 audit set [27, 67, 91] assembled by iterative search and snowballing (Appendix A): (i) a 7-tuple skill plus a library-level operator vocabulary (§2); (ii) an eight-stage lifecycle architecture and an eight-family taxonomy (§§3–4); and (iii) five evidence-graded patterns (§5) and a five-item reporting checklist (§10). We treat classical options and hierarchical RL [4, 14, 31, 47, 56] as background anchors rather than corpus members, and discuss memory-only agents and tool-use systems as boundary cases.

2 A Formal Lens for Dynamic Skills

The phrase "agent skill" covers at least six structurally different artifacts: executable code (Voyager, SAGE), SKILL.md packages (MetaClaw, Memento), NL heuristics (ERL, EvolveR), parametric adapters (SkillsCrafter, SKILL0), memory/trajectory items (SimpleMem, MUSE), and capability labels (Co-Evolving-Agents) [2, 30, 42, 45, 60, 62, 63, 69, 74, 78, 97]. We separate *terminology* (Appendix B) from *formalism*: the formalism extends the options-style 4-tuple from Jiang et al. [27] with the minimum machinery needed to describe edits, verification, lineage, and library-level change.

From the static 4-tuple to the dynamic 7-tuple. Sutton et al. [56] model an option as $\langle I, \pi, \beta \rangle$, and Jiang et al. [27] reinterpret this for LLM agents as $\mathcal{S} = \langle C, \pi, T, R \rangle$ with applicability C, executable policy π, termination T, and reusable interface R. This is adequate for static libraries, but a dynamic library requires three additional components:

$$\mathcal{S}_t = \langle C_t, \pi_t, T_t, R_t, \varphi_t, \nu_t, \prec_t \rangle. \tag{1}$$

Here φ_t is the candidate-generation rule (deterministic in some systems, a proposal kernel $q_t(\mathcal{S}' \mid \mathcal{S}_t, u_t)$ in mutation-based systems such as CODE-SHARP [6]); ν_t is the admission predicate that decides whether a candidate enters $\mathcal{L}_{t+1}$ (a unit test in LATM, a grounded rollout in SkillWeaver/EvoSkill, an ensemble judge in Trace2Skill/AgentSkillOS, a symbolic audit in ASG-SI, or a Bayesian utility prior in CODE-SHARP); and $\prec_t$ is a partial order over versions that supports rollback (PSN's 20%-success rollback gate [53]), maturity gating, and provenance. Static libraries are the limiting case where no update trigger produces an admitted edit, so $\mathcal{L}_{t+1} = \mathcal{L}_t$ except for exogenous releases; an unconditional $\nu_t \equiv 1$ with repeated Add would instead describe an unfiltered append-only store. A worked end-to-end instantiation on AutoRefine is in Appendix C.

Library-level dynamics. Write the library as $\mathcal{L}_t = \{\mathcal{S}_t^{(1)}, \ldots, \mathcal{S}_t^{(N_t)}\} \cup \mathcal{M}_t$ with metadata $\mathcal{M}_t$ (invocation statistics, call graphs, maturity labels). The transition is

$$\mathcal{L}_{t+1} = \mathrm{Apply}\big(u_t(\tau_t, r_t),\ \mathcal{L}_t\big), \tag{2}$$

where τ_t is an evolution trigger (timer, task end, failure event, user edit), r_t is a learning signal (reward, NL critique, judge score, cross-user aggregate, teacher), and u_t is an operator choice or short operator sequence drawn from the ten-operator vocabulary

$$\begin{aligned} O = \{&\textsc{Add}, \textsc{Refine}, \textsc{Merge}, \textsc{Split}, \textsc{Prune}, \\ &\textsc{Distill}, \textsc{Abstract}, \textsc{Compose}, \\ &\textsc{Rewrite}, \textsc{Rerank}\}. \end{aligned}$$

The operator semantics are fixed throughout the paper: Add inserts; Refine edits content without changing the interface; Merge combines; Split factors; Prune removes or quarantines; Distill compresses trajectories into a skill; Abstract lifts a concrete procedure to a template; Compose chains skills; Rewrite changes the body and possibly the interface; Rerank changes retrieval priors. Almost no method implements all ten; the supported subset is one of the clearest taxonomic fingerprints (Appendix M).

What the lens separates. The 7-tuple separates skills from *tools* and from *memory*. A tool is externally supplied with a stable API; a skill is an artifact whose lifecycle is managed by the agent. The same Python function or MCP endpoint is a tool in one system and a skill in another if the agent proposes it, verifies it for admission, stores lineage, and later maintains or transfers it. Memory items are passively retrievable context; skills are invocable, editable, and admitted. The distinction is operational: what matters is whether the artifact participates in $\mathcal{L}_t \to \mathcal{L}_{t+1}$.

Feasible $(\varphi, \nu, \prec)$ depends on artifact type. Equation 1 is an unconstrained schema; in practice the artifact type narrows which gates and edit operators are realistic. Executable code admits machine-checkable ν (unit tests, sandbox rollouts, contracts) and per-line φ, but cross-version $\prec$ requires explicit version control because identical syntax is compatible with incompatible runtime assumptions. Natural-language heuristics admit cheap φ but only indirect ν (downstream task success, judge agreement), which is why retrieval-time effects dominate the regularities for this family. SKILL.md packages combine both: L1 metadata supports routing, L2 prose supports judge-style ν, and L3 attachments restore execution-grounded admission when present. Parametric skills move φ and ν onto a training timescale and trade per-skill auditability for amortized inference; capability labels lack a robust body against which φ can be validated, which is why Table 5 (Appendix B) treats them as a boundary case. The artifact type therefore determines which subset of the

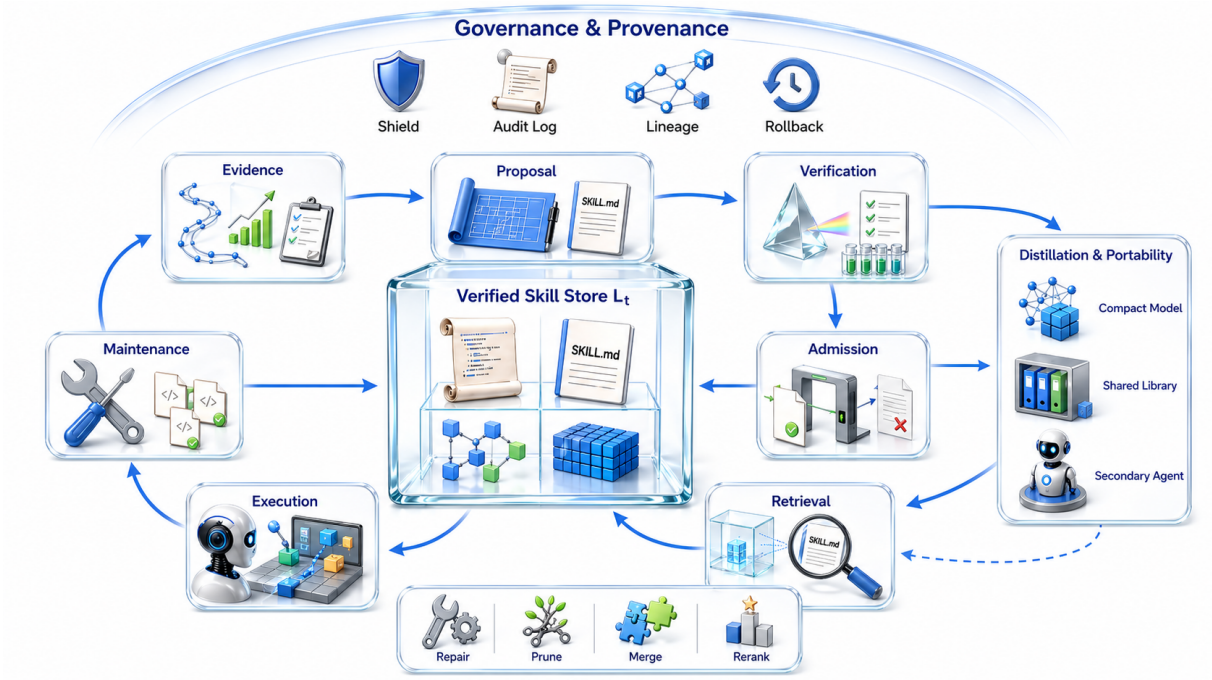

Figure 1. Dynamic skill systems as lifecycle-managed artifact stores. Interaction evidence drives proposal and verification; admitted artifacts enter an evolving store $\mathcal{L}_t$, where retrieval and execution create more evidence. Maintenance repairs, prunes, merges, or reranks the store; governance and provenance wrap the lifecycle; distillation and portability form a slower side loop.

operator vocabulary is even available, before any choice about which subset a given system implements.

3 The Lifecycle Architecture

A dynamic skill system observes interaction evidence, proposes a skill or library edit, verifies the candidate, admits it into a storage topology, retrieves or composes it at future invocation, maintains it as the library ages, and records enough provenance to support rollback, transfer, and governance. We decompose this into eight recurring stages (Figure 1).

The eight stages are: **evidence acquisition** (trajectories, rewards, failure traces, user edits, cross-user signals, external resources); **proposal** (turn evidence into a candidate artifact or edit); **verification & admission** (decide whether the candidate enters the library, and as mature, tentative, quarantined, or rejected); **organization** (flat index, hierarchy, DAG, invocation graph, ontology, dual store, parametric subspace); **retrieval & composition** (decide which artifacts influence the next action); **maintenance & repair** (Prune, Merge, Split, Rerank, Refine, Rewrite); **distillation & portability** (move knowledge between artifacts, weights, agents, users, or domains); and **governance & provenance** (lineage, authorship, safety checks, release state, rollback handles).

The order is not a waterfall: many systems loop between proposal and verification, retrieve before maintaining, or distill only periodically. Each stage asks a different design question, and a system that performs strong retrieval has not necessarily solved admission. The most important boundary in the lifecycle is between *candidate* and *admitted* state. Execution-grounded systems (SkillWeaver, EvoSkill, SkillFoundry) place that gate at write time; maintenance-heavy systems (AutoRefine, AgentSkillOS) re-run it as the library ages; rollback systems (PSN) treat admission as tentative until recent-task utility remains stable [3, 34, 49, 52, 53, 92]. Without operator-level provenance, dynamic skills become append-only memories with a more polished file format. A per-stage operator-and-failure decomposition is in Appendix F.

Staged admission is the rule, not the exception. The strongest dynamic-skill systems do not treat admission as a binary accept/reject. A candidate enters a state with maturity and rollback semantics: it may be *tentative* (admitted under a probationary window where downstream success is monitored), *quarantined* (admitted but blocked from retrieval until further evidence accumulates), *promoted* (graduated from tentative to mature after successful invocations), *demoted* (kept in the library but ranked below alternatives), or *rolled back* (reverted because the lineage

relation $\prec$ identified a regression). PSN's 20%-success rollback gate [53], AGENTIC-PROPOSING's maturity-gated ensemble admission [29], and AGENTSKILLOS's capability-tree audits [34] are three concrete instantiations. This is why the verifier v in Equation 1 acts at admission rather than at the moment of the edit: the same proposal can be tentatively admitted under a weaker probationary verifier and re-evaluated under a stricter audit before promotion.

A worked transition. A concrete AUTOREFINE-style maintenance cycle [49] is illustrative. At time t, the agent runs a TravelPlanner trace and records evidence $\tau_t = \text{PERIODIC}$, r_t a critique plus utility log. The update rule chooses an operator vector

$$\begin{aligned}\vec{u}_t = \{&(\text{REFINE}, \text{patch step}),\\ &(\text{MERGE}, \text{combine duplicates}),\\ &(\text{PRUNE}, \text{quarantine low-utility})\}.\end{aligned}$$

Each realized edit yields a candidate $\mathcal{S}^*$. The admission predicate v_t runs a rubric judge plus a replay test; if the candidate passes, $\mathcal{S}^*$ enters $\mathcal{L}_{t+1}$ and $\prec_{t+1}$ records that it supersedes earlier artifacts; if it fails, $\mathcal{L}_{t+1}$ retains the prior version or marks the candidate for review. The transition is not a single append: it is a gated composition of REFINE, MERGE, and PRUNE over a versioned store. This is what AUTOREFINE's maintenance-off ablation removes, and why the resulting library grows from 28 to 126 skills while utilization falls from 0.71 to 0.08 [49].

What counts as dynamic. We use *dynamic* for systems with a non-trivial library transition $\mathcal{L}_t \to \mathcal{L}_{t+1}$, not for any system that retrieves a skill at inference time. Retrieval changes the context; dynamic update changes the store. A method can therefore be skill-using without being dynamic, and it can be dynamic in a narrow way if it implements only one transition such as ADD after each task. Lifecycle maturity increases as systems add admission, maintenance, lineage, governance, and two-timescale consolidation. The taxonomy in §4 asks which subsets of those commitments each surveyed system actually implements.

4 A Lifecycle Taxonomy

We organize the corpus along two axes that emerge from the lifecycle: which artifact a family edits, and which subset of the operator vocabulary it actually implements. Table 1 groups the 94 systems into eight families. The full per-system master coding sheet (Appendix D) records seven coding fields per system; we emphasize here that they are *not orthogonal axes*—artifact type constrains verification, storage topology constrains maintenance, and update locus constrains the available signal. Three couplings summarize the most visible coded patterns rather than proven causal laws.

The three couplings. *Artifact–verifier:* executable code supports tests and rollouts; NL lessons and capability labels rely on indirect downstream or judge-based checks; parametric skills move verification onto a training timescale. *Storage–maintenance:* flat stores make ADD cheap but MERGE and PRUNE expensive; graphs and hierarchies expose structure but require careful rollback. *Trigger–signal:* task-end retrospection naturally supplies critique, RL loops supply reward, user edits supply ownership constraints, and deployment registries supply aggregate utility. Appendix E reports aggregate operator counts from the representative coding sheet; they support these couplings qualitatively, but the current literature is too heterogeneous for causal estimates.

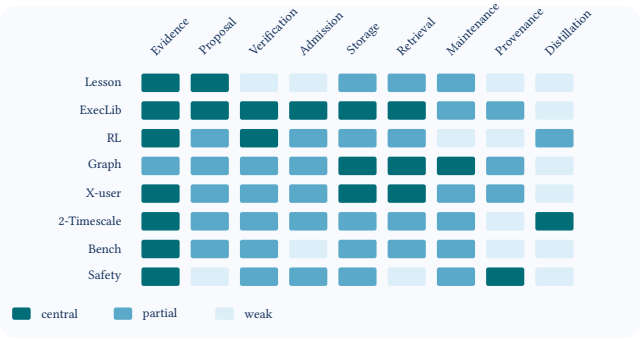

Figure 2. Lifecycle coverage across system families. Cell intensity summarizes how centrally a family implements each lifecycle stage. Executable libraries concentrate around proposal/verification/admission; graph systems around storage/maintenance; two-timescale systems around distillation; safety/governance around provenance.

The heatmap in Figure 2 makes the central observation visible: families differ less in whether they "use skills" than in which lifecycle stages they actually cover. Once skills are packaged for registries, scanners, package managers, and repository-context checks become part of the same taxonomy as routing and maintenance.

5 Mechanisms and Five Evidence-Graded Patterns

Across the corpus, four mechanism choices are load-bearing: which edits a system can make, how candidates are admitted, how the resulting store is organized and retrieved, and when external skills are consolidated into slower stores. The choices are coupled—a wide edit repertoire requires stronger verification, and distillation is useful only when the fast-loop library is selective enough to give a clean training signal. Within those mechanism families, the corpus surfaces five patterns with different evidence grades (Table 2). We reserve stronger language for patterns backed by controlled ablations and use weaker language where the evidence is benchmark-specific or architectural.

Table 1. Primary lifecycle taxonomy of dynamic skill systems. Families differ mainly in which lifecycle stages they make cheap, verified, or governable. Operator footprint is drawn from the ten-operator vocabulary of Equation 2.

Family	Dominant lifecycle stages	Operator footprint	Assurance bottleneck	Representative systems / residual failure
Retrospective lesson induction	evidence → proposal → retrieval; task-end updates	Add, Refine, Abstract, Distill	indirect via self-critique or downstream success	ERL, RetroAgent, EvolveR, MUSE, XSkill, SAGER; lesson drift, visual mismatch
Executable PPVH libraries	proposal → verify → admit → execute	Add, Refine, Prune, Compose	verifier coverage outside sampled contracts	SkillWeaver, EvoSkill, CoEvoSkills, SkillCraft, SkillFoundry, SkillForge; verifier narrowness
Skill-aware RL	evidence/proposal inside RL; optional slow distillation	Add, Refine, Rerank, Distill	verifier hacking, reward–skill mismatch	SAGE, SkillRL, AgentEvolver, CODE-SHARP, Tool-R0; reward misspecification
Graph / hierarchy systems	organization → maintenance → retrieval/composition	Split, Merge, Compose, Rerank, Prune	rollback and structural-validation cost	PSN, AgentSkillOS, SkillX, Graph of Skills, GraSP; structure decay
Cross-user / registry sharing	portability → retrieval → governance across users/agents	Add, Refine, Merge, Rerank, Prune	ownership, privacy, retrieval quality, and compatibility checks	SkillClaw, AutoSkill, SkillFlow-2025, AgentDevel, EvoAgent; dominant-user bias, leakage, retrieval drift
Two-timescale internalization	fast external store → slow consolidation	Distill, Abstract, Prune, Rerank	quality of the distillation window and probe	MetaClaw, K2-Agent, SKILL0, SkillsCrafter, SELAUR; parametric collapse
Lifecycle benchmarks	measure proposal, patching, retrieval, repair	measures Add, Refine, Prune, usage, quality gaps	evaluator realism, global-library validity	SkillFlow-Bench, Wild-Skills, SkillsBench [36], SkillLearnBench; reveal failures, do not solve them
Governance / safety	admission/invocation checks plus provenance and release review	Prune, quarantine, audit, rollback	attack coverage, defended-system integration	ASG-SI, ClawSafety, SkillSieve, Skilldex, MedSkillAudit; sparse integration with live systems

R1: Admission gates matter. The supported claim is not merely that filtering is useful; it is that *where and how* the gate fires changes future library state. CoEvoSkills [86] provides the cleanest verifier ablation: removing the surrogate verifier drops SkillsBench pass rate from 71.1% to 41.1% (Table B1, a 30-point effect). EvoSkill's Pareto-front selection over held-out validation yields +7.3 and +12.1 absolute points on its two primary benchmarks [3]. SkillWeaver's practice+verify ablation drops below the no-skill baseline on hard web tasks [92]. Maintenance-heavy systems (AutoRefine, AgentSkillOS) report similar erosion when the gate is disabled [34, 49]. The exception is case-based memory (SimpleMem, parts of MUSE [42, 78]), where admission is closer to relevance filtering than quality control.

R2: Verifier quality is decisive in skill-aware RL. The distinctive mechanism is persistence: a verifier does not only score one rollout; it decides which artifacts enter the future action space. The 30-point CoEvoSkills effect; CODE-SHARP's 24.30% → 41.02% gain under a learned gate [6]; Agentic-Proposing's verifier ensemble improving problem validity from 68.7% to 82.3% [29]; Co-Evolving's hard-negative gain [30]; and SELAUR's uncertainty-aware reward shaping [85] all point to verifier design as one of the most load-bearing engineering choices, often comparable in effect to the RL algorithm itself. In execution-verified libraries, additional rollouts may have diminishing returns relative to Prune and Rerank.

R3: Flat retrieval can drop at moderate library scale. The strongest evidence is a controlled Single-Agent-Skills size sweep; the broader claim is a skill-library instance of a familiar retrieval signal-to-noise problem rather than a new universal law. Single-Agent-Skills [35] gives the clearest controlled curve (Figure 3); Wild-Skills [44] shows a related failure under realistic distractors and 34K-pool retrieval; SkillRouter [95] requires full-text retrieve-and-rerank at 80K scale; Graph of Skills [41] shows dependency-aware retrieval preserving reward up to 2K skills. Hierarchy, DAG, and ontology storage push the knee rightward; the literature has not shown general removal.

R4: Maintenance appears load-bearing after growth. This is the most domain-specific pattern but also the least settled as a cross-paper causal claim. The cleanest direct evidence is AutoRefine's TravelPlanner maintenance-off ablation: final pass rate drops from 35.6% to 31.1%, the repository grows 4.5× from 28 to 126 skills, and utilization falls from 0.71 to 0.08 (Figures 2–3 of [49]). AgentSkillOS attributes scaling to periodic capability-tree audits; Wild-Skills and EffiSkill [65] show utility-based filtering beats added admission; PSN's maturity gating loses its advantage when disabled; SkillFlow-Bench [91] adds that weaker settings often fail by skill inflation rather than inability to write skills. We treat maintenance as a well-motivated hypothesis with one strong ablation plus convergent support, not as a settled law.

R5: Write-time abstraction often beats read-time alone. Retrospective induction during authoring usually

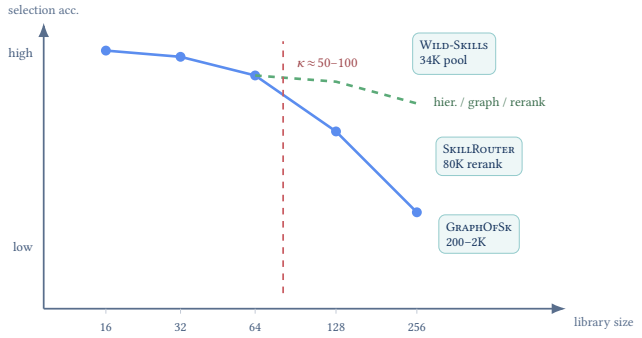

Figure 3. Retrieval-scaling evidence behind R3. The blue curve renders the controlled Single-Agent-Skills size-sweep pattern; the green dashed line summarizes the hierarchy/graph/rerank mitigation family. Wild-Skills, SkillRouter, and Graph of Skills push the failure rightward but are not yet comparable under one shared harness.

outperforms read-time summarization. Trace2Skill's prevalence-weighted consolidation beats a read-time summarizer baseline [48]. CASCADE's write-time distillation beats retrieval-plus-rerank [24]. SimpleMem's Table 5 shows that removing write-time semantic compression drops LoCoMo F1 from 43.24 to 31.29. XSkill's Table 3 ablation shows that removing the Experience Manager drops VisualToolBench Average@4 by 4.09 and removing the Skill Manager drops it by 3.62, while read-time decomposition/adaptation ablations are smaller [26]. The result is cleanest under reasonably stationary tasks; non-stationary deployments may need read-time adaptation layered on top.

Cross-cutting observation: write-time discipline. R1, R2, R4, and R5 are all forms of *write-time discipline*: what enters the library, under what gate, and with what abstraction. R3 and the focused-library effect (Appendix L) argue that smaller can be better through two distinct mechanisms—retrieval-resolution collapse and distractor-load saturation—and the response is the same: invest in Prune, Merge, and Rerank rather than only Add. The recurrent failure of monotonic-growth systems is therefore not a coincidence; it is the absence of a negative operator. Lifecycle benchmarks repeatedly warn that skill *usage* is not skill *utility*: SkillFlow-Bench reports Claude Opus 4.6 improving from 62.65% to 71.08% task completion under skill evolution, while Kimi K2.5 gains only +0.60 points despite 66.87% skill use, and GPT 5.3 Codex regresses by 6.02 points [91]. The same gap appears under realistic retrieval: Wild-Skills [44] shows that curated skills lose much of their advantage once distractors are added at corpus scale, and that utility-based filtering can outperform increased admission alone.

Convergent pattern: weaker backbones gain disproportionately. A separate stylized fact is that several studies report larger *relative* gains for weaker base models than for stronger ones. SkillWeaver [92] closes a larger fraction of the capability gap on weaker web-agent backbones; MetaClaw [74] reports larger relative two-timescale gains on weaker backbones in its setting; EvoSkill [3] reports analogous gap compression; and Agentic-Proposing's headline result is on a 30B model, with a smaller relative lift on the frontier teacher. We do not promote this to a regularity here because the cross-paper evidence varies in harness, context budget, and gain definition, but the pattern is convergent enough to make a deployment-facing prediction: dynamic-skill machinery is most likely to pay off where weaker backbones or tight inference budgets are the binding constraint, not primarily where frontier models are already strong. XSkill's Qwen transfer is the caution [26]: transferred knowledge can raise Pass@4 by encouraging exploration while lowering Average@4, so the gain is sometimes a redistribution of rollout quality rather than a uniform improvement.

Operator economy is set by the substrate. A practical consequence of the storage–maintenance coupling is that infrastructure choices predetermine which operators a system can run continuously. Flat embedding indices (EvoSkill, SkillWeaver, Agentic-Proposing) make Add cheap but Merge and global Prune expensive; hierarchical stores (AgentSkillOS, SkillX, Corpus2Skill) make Split cheap but subtree Prune risky; DAGs (CODE-SHARP, PSN, Graph of Skills, GraSP) make Compose auditable but Prune structurally expensive because descendants may depend on a removed node; ontologies (SkillOrchestra) make category inheritance cheap but category-changing Refine expensive. Three pipeline architectures recur: *inline editing* (PSN, CODE-SHARP) requires an explicit version store for rollback; *log-and-apply* (AutoRefine, AgentSkillOS) stages the edit and commits only after admission; *shadow execution* (MetaClaw, SKILL0) evolves a shadow library while serving from the main one and cuts over during a distillation window. The full operator-economy table is in Appendix J.

6 The Evaluation Gap

A benchmark that reports only endpoint task success hides the phenomena that define dynamic skills: skill inflation, incorrect-skill drift, maintenance-off collapse, retrieval knees, and usage without utility. Four metric families dominate today: terminal pass@k, sample efficiency to threshold, library-size and retrieval stress tests, and lifecycle metrics (final skill count, file-kind composition, skill-use rate, generated-skill quality, and the gap between use and improvement) [26, 35, 44, 91, 95, 96].

Table 2. Five evidence-graded workshop patterns. Evidence is strongest for within-paper ablations and weakest for architectural corroboration. Methods use different backbones, harnesses, and benchmarks, so cross-paper numbers are convergent rather than pooled.

Pattern	Grade	Evidence pattern	Design implication
R1. Admission gates matter	A/B	CoEvoSkills surrogate-verifier ablation 71.1% → 41.1%; EvoSkill +7.3/+12.1; SkillWeaver ablation drops below no-skill baseline; AutoRefine, AgentSkillOS maintenance-gate ablations	Report admission policy, rejected candidates, verifier failure modes
R2. Verifier quality is decisive in skill-aware RL	A/B	CODE-SHARP 24.30% → 41.02%; Agentic-Proposing 68.7% → 82.3% via ensemble; Co-Evolving hard-negative gain; SELAUR uncertainty-aware shaping	Treat verifier design as part of the method, not an implementation detail
R3. Flat retrieval can drop at moderate scale	B/C	Single-Agent-Skills controlled 64–128-skill curve; Wild-Skills 34K-pool degradation; SkillRouter 80K full-text rerank; Graph of Skills 200–2K dependency retrieval	Plot performance vs. library size; report storage topology
R4. Maintenance appears load-bearing after growth	B	AutoRefine TravelPlanner: 35.6 → 31.1 final pass, 28→126 skills, 0.71 → 0.08 utilization; convergent support from AgentSkillOS, Wild-Skills, EffiSkill, PSN, SkillFlow-Bench	Measure repair quality and post-maintenance utility; report negative-operator velocity
R5. Write-time abstraction often beats read-time alone	A/B	SimpleMem 43.24 → 31.29 (Table 5); XSkill −4.09/−3.62 Avg@4 (Table 3); Trace2Skill; CASCADE	Place Abstract/Distill at write time, not only at read time

Two quantities remain under-reported. *Operator velocity*—counts of Add, Refine, Merge, Prune, and Rerank per task or per dollar—is missing in almost every method paper, which is why the velocity–soundness tradeoff between PSN, CODE-SHARP, SkillMOO, and Bilevel-MCTS [6, 18, 22, 53] cannot be compared quantitatively. *Repair quality*—whether a later patch actually corrects a faulty abstraction—is exposed qualitatively by SkillFlow-Bench and SkillForge [43] but lacks a standard scalar. SkillFlow-Bench's 166-task family-reset protocol is the most lifecycle-aligned benchmark today, but its family-local design leaves open how a single global library behaves under heterogeneous workflows.

Cross-paper comparability is hard. Benchmark progress does not yet make the literature head-to-head comparable. *Cross-operator comparison* lacks operator velocities. *Cross-library-size comparison* lacks smooth size sweeps—only Single-Agent-Skills [35] and Graph of Skills [41] report controlled curves. *Cross-backbone comparison* is confounded by model, harness, context length, and tool surface, even in useful early transfer studies such as CoEvoSkills and XSkill. *Cross-task-distribution comparison* remains immature: LifelongAgentBench [93] and ProEvolve [33] provide adjacent protocols, but most deployment papers still use bespoke task streams. A trajectory-aware protocol should report the library as a time series, with four elements: performance, skill count, and retrieval quality at a grid of task indices or library sizes; operator velocities; a drift schedule or family-transfer condition; and a maintenance-off or repair-off ablation. Appendix I records the full benchmark map.

7 Safety as a Lifecycle Stage

The April 2026 safety wave turns dynamic skills from a prompt-injection concern into a software-supply-chain problem. Skill artifacts combine natural language, code, configuration, sometimes learned models, and social provenance, so admission control, provenance, sandboxing, and release governance must be lifecycle stages rather than after-the-fact filters. Eight surfaces (Appendix K) are specific to dynamic skill stores; we summarize four families here.

Injection and supply-chain. ClawSafety [67] reports malicious skill files as the highest-ASR injection vector (69.4% averaged over backbones); Supply-Chain-Poisoning's DDIPE attacks bypass scanners 11.6–33.5% of the time across 1,070 adversarial skills [50]; BadSkill bundles backdoored models in skills with 97.5–99.5% ASR while preserving benign accuracy [58].

Harmful-but-valid skills and scanner limits. Harmful-SkillBench finds 4,858 harmful skills among 98,440 collected, and even implicit invocation can raise harm scores [28]. SkillSieve filters 86% of skills at zero API cost and reaches 0.800 F1 at $0.006 per skill [21]; Malicious-or-Not shows scanner-only classification overstates risk—adding repository context reduces 2,887 scanner-flagged combinations to 0.52% remaining as flagged [19].

Credential leakage and skill theft. Credential Leakage reports 76.3% of credential-exposure issues require joint NL+code analysis (debug logging accounts for 73.5% of issues, because stdout is fed back to the LLM) [12]. SkillStealing demonstrates black-box extraction of proprietary skill behavior with few interactions [66]. *Domain release.* MedSkillAudit [20] finds 57.3% of 75 medical research skills fall below their Limited Release threshold (ICC= 0.449 vs. expert consensus). *Attribution and composition.* Maintenance operators delete, merge, split, and rewrite skills; without operator-level provenance, later audits cannot reconstruct which artifact contributed to a decision, and individually acceptable skills can be harmful when chained [23]. A defended deployment needs at least six governance primitives—an admission-time

scanner combining static checks, LLM/rubric analysis, sandboxing, and repository provenance; a package manifest for dependencies, permissions, model artifacts, and allowed tools; a runtime containment layer for stdout, filesystem, and network channels; an operator-granular provenance log for Add, Refine, Merge, Split, and Prune; a policy and domain release gate; and a market governance layer for ownership, takedown, and remediation—each tied to a specific lifecycle stage. Table 3 summarizes the eight surfaces with their quantitative anchors; per-surface detail is in Appendix K.

Table 3. Eight safety surfaces specific to dynamic-skill stores, with quantitative anchors.

Surface	Quantitative anchor
1. Skill-file injection	ClawSafety 69.4% ASR (highest of three vectors) [67]
2. Supply-chain poisoning	Supply-Chain 11.6–33.5% bypass over 1,070 skills; BadSkill 97.5–99.5% ASR with backdoored bundled models [50, 58]
3. Harmful-but-valid skills	HarmfulSkillBench 4,858 of 98,440 skills (4.93%) raise harm even under implicit invocation [28]
4. Credential leakage	Credential Leakage 76.3% require joint NL+code; debug-log channel accounts for 73.5% of issues [12]
5. Scanner limits / FP	SkillSieve 0.800 F1 / $0.006 per skill; Mal-or-Not drops 2,887 flagged skills to 0.52% with repository context [19, 21]
6. Ownership / theft	SkillStealing few-query black-box extraction [66]
7. Domain release	MedSkillAudit 57.3% of 75 skills below Limited Release; ICC=0.449 vs. experts [20]
8. Attribution / composition	ASG-SI graph-level audit; SkillOrchestra typed composition (partial) [23, 61]

8 Open Problems

The five evidence-graded patterns constrain the next research agenda. We state five open problems below, each with an evidence-basis tag and a concrete first experiment; Appendix L adds three further infrastructure-level problems with the same structure.

1. **Compositional verifiers and grounded maintenance schedules.** (*extrapolated.*) Probe-set rollback validation (PSN) and learned judges (CODE-SHARP) cover different defect classes; no method composes them. Maintenance schedules are engineering defaults; deriving when to invoke Prune, Merge, or Distill from monitored quantities (admission rate, drift rate, retrieval entropy) is open. *First experiment:* replace one CODE-SHARP mutation with a PSN-style rollback-gated variant and measure the velocity-vs-admission frontier; derive an optimal maintenance frequency from a cost model and compare against the periodic-distillation defaults of MetaClaw and SKILL0.
2. **Admission under non-stationary task distributions.** (*extrapolated.*) Yesterday's verifier becomes a poor predictor of today's utility; existing systems (SkillClaw, AutoSkill, MetaClaw) evict but do not adapt the verifier itself. *First experiment:* wrap a skill-aware RL verifier in a utility tracker that shifts the admission threshold from downstream success, then test on a planted distribution-shift benchmark; the obstacle is that no such benchmark currently exists.
3. **Parametric collapse under repeated distillation.** (*speculative.*) No two-timescale system runs long enough to test whether each distillation window biases proposals toward what the model just absorbed; the missing evidence is long-horizon proposal diversity. *First experiment:* run any two-timescale system for 10× its reported distillation windows and track proposal diversity; the obstacle is compute budget for long horizons.
4. **Retrieval and portability beyond the moderate-size knee.** (*measured at moderate scale; extrapolated to marketplace scale.*) Hierarchy, DAG, ontology, and routing push the 64–128-skill knee rightward but do not prove general removal at 10^4–10^6 skills. Cross-library portability adds a parallel question: when is a skill authored on agent A safe and useful on agent B? CoEvoSkills reports broad cross-model transfer; XSkill reports mixed multimodal transfer. A compatibility theory should predict outcomes before evaluation. *First experiment:* evaluate learned routing, graph retrieval, skill compilation, and hybrid parametric–retrieval at 10^4–10^6 skills; extend transfer studies into a controlled cross-product of author backbone, receiving backbone, harness, and tool API.
5. **Honest dynamic benchmarks with provenance.** (*measured benchmark gaps; extrapolated integration.*) SkillFlow-Bench captures temporal patching; Wild-Skills captures retrieval realism. The open problem is to combine them while reporting global-library trajectory, operator velocity, and usage-vs-utility, all under operator-level provenance compatible with 10^2–10^3 fast-loop edits per day [12, 19, 21, 23]. *First experiment:* combine the SkillFlow-Bench sequential-patch protocol with Wild-Skills retrieval realism and report all three quantities under ASG-SI-style audited provenance; the obstacle is standardizing a shared global-library protocol across task families.

The first three problems improve algorithms inside existing mechanism families and are tractable for a single team. The last two require community infrastructure for portability, governance, and trajectory-aware benchmarks. Method papers and infrastructure papers should therefore be judged together: dynamic skills will not mature if algorithms improve inside isolated libraries while portability, provenance, and benchmark realism lag behind.

9 Limitations

The corpus is frozen at the April 24, 2026 cutoff and assembled by iterative search and snowballing

rather than a single PRISMA-style query; false negatives are likely for unpublished systems, product documentation, non-English papers, and papers using "tool/memory/workflow/curriculum" terminology without the word "skill". The evidence is heterogeneous and still maturing: most primary systems are 2025–2026 preprints, and few have independent replications. A verifier ablation is stronger than a single benchmark score; a registry-scale measurement is stronger evidence for a deployment surface than for a remedy. The lifecycle framing fits artifact-store systems best; purely parametric methods fold proposal/verification/admission into a training loop, and memory-only methods may not form an invocable interface. The operator vocabulary is intentionally coarse and taxonomic rather than a mathematical algebra: it does not model edit dependencies (a Refine that triggers downstream Prune, a graph rewrite that changes both retrieval and composition semantics, or a stochastic proposal distribution that emits many candidates before one is admitted) or hierarchical edits where one library transition triggers another. Most claims are architectural rather than causal, and we deliberately provide no quantitative cross-paper leaderboard—backbone, harness, context budget, and evaluator vary too much for pooled effect sizes to be meaningful without a shared harness.

10 Reporting Standard and Conclusion

Externalizing procedural knowledge turns the skill library into part of the learning system: it has state, update rules, selection pressure, maintenance costs, and safety constraints. The lifecycle view, the time index, the write-time discipline pattern, and operator pluralism are now well supported in the literature; the claim that scaling is solved is not. Hierarchy, DAG, ontology, routing, and orchestration push the retrieval knee rightward without removing it; cross-library portability, provenance, and benchmark honesty remain open.

For the next wave to become comparable, dynamic-skill papers should report the five quantities in Table 4. Method papers should identify which lifecycle stages they implement and which they only assume; report operator velocities and library trajectories rather than only final task success; include repair-, maintenance-, or admission-off ablations when those stages are part of the method; distinguish skill usage from skill utility; and (for infrastructure or safety substrates) state which operators their substrate makes cheap, expensive, blocked, or auditable. Safety papers should additionally evaluate admission, composition, and provenance surfaces rather than treating skills as ordinary prompt files. If these quantities become standard, the next survey can make quantitative cross-method claims that this one deliberately avoids.

Table 4. Five-item reporting checklist for dynamic-skill papers. Each item corresponds to a reporting gap observed in the surveyed corpus.

Item	What to report	Minimum decision criterion
1. Lifecycle stages	Realized vs. assumed stages: evidence, proposal, verification, admission, storage, retrieval, maintenance, distillation, governance	A reader can locate the claimed library transition
2. Operator velocities	Counts and costs for Add, Refine, Merge, Prune, Rerank; $\lvert\mathcal{L}_t\rvert$ over time	At least one trajectory plot or table, not only final size
3. Gate / repair ablations	Admission-, maintenance-, repair-, or verifier-off comparisons	Ablation targets the lifecycle stage claimed as contribution
4. Use vs. utility	Skill-use rate, downstream gain, stale-skill and wrong-skill rates	A used skill must be separable from a useful skill
5. Substrate economy	Which operators the storage/registry/runtime makes cheap, expensive, blocked, or auditable	Operator costs stated qualitatively or quantitatively

The checklist is a measurement standard rather than an architecture. It asks each paper to expose the transition it claims to improve: how many candidates were proposed, rejected, admitted, used, repaired, made stale, or removed; which verifier covered which admission boundary; which provenance survived maintenance or distillation; and which operator caused the measured gain. It also gives a negative criterion. A system that only appends skills is incomplete unless it controls retrieval pollution, stale entries, and unsafe candidates; a system that only routes over a fixed store is not yet dynamic unless the store can change; and a system that claims lifelong improvement should expose the time series of library states.

Our position is conservative. Dynamic-skill research should not claim that agents become open-ended simply because they can write skills. It should claim progress when the store changes in a measured, verified, and inspectable way. That framing leaves room for code libraries, SKILL.md packages, workflow graphs, registries, and parametric consolidation while keeping the evaluation target stable: the scientific object is the verified transition $\mathcal{L}_t \rightarrow \mathcal{L}_{t+1}$ and the evidence that justifies it, not the presence of a directory named "skills."

Acknowledgments

This research was supported in part by the National Institute of Standards and Technology under Federal Award ID 60NANB24D231 and by Carnegie Mellon University's AI Measurement Science and Engineering Center (AIMSEC).

References

[1] Emre Can Acikgoz, Cheng Qian, Jonas Hübotter, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. 2026. Tool-R0: Self-Evolving LLM Agents for Tool-Learning from Zero Data. *arXiv:2602.21320* (2026).

[2] Marc-Antoine Allard, Arnaud Teinturier, Victor Xing, and Gautier Viaud. 2026. Experiential Reflective Learning for Self-Improving LLM Agents. *arXiv:2603.24639* (2026).

[3] Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. EvoSkill: Automated Skill Discovery for Multi-Agent Systems. *arXiv:2603.02766* (2026).

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The Option-Critic Architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31. 1726–1734. doi:10.1609/aaai.v31i1.10916

[5] Shuzhen Bi, Mengsong Wu, Hao Hao, Keqian Li, Wentao Liu, Siyu Song, Hongbo Zhao, and Aimin Zhou. 2026. Automating Skill Acquisition through Large-Scale Mining of Open-Source Agentic Repositories: A Framework for Multi-Agent Procedural Knowledge Extraction. *arXiv:2603.11808* (2026).

[6] Richard Bornemann, Pierluigi Vito Amadori, and Antoine Cully. 2026. CODE-SHARP: Continuous Open-ended Discovery and Evolution of Skills as Hierarchical Reward Programs. *arXiv:2602.10085* (2026).

[7] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large Language Models as Tool Makers. *arXiv:2305.17126* (2023).

[8] Le Chen, Erhu Feng, Yubin Xia, and Haibo Chen. 2026. SkVM: Revisiting Language VM for Skills across Heterogenous LLMs and Harnesses. *arXiv:2604.03088* (2026).

[9] Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, Ning Miao, Siyang Gao, Cong Lu, Manling Li, Junxian He, and Yee Whye Teh. 2026. SkillCraft: Can LLM Agents Learn to Use Tools Skillfully? *arXiv:2603.00718* (2026).

[10] Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Leon Xu, Suzhen Zheng, Hao Fan, Pashmina Cameron, Justin Wagle, and Kazuhito Koishida. 2026. CUA-Skill: Develop Skills for Computer Using Agent. *arXiv:2601.21123* (2026).

[11] Xiaoyin Chen, Canwen Xu, Yite Wang, Boyi Liu, Zhewei Yao, and Yuxiong He. 2026. Learning to Self-Evolve. *arXiv:2603.18620* (2026).

[12] Zhihao Chen, Ying Zhang, Yi Liu, Gelei Deng, Yuekang Li, Yanjun Zhang, Jianting Ning, Leo Zhang, Lei Ma, and Zhiqiang Li. 2026. Credential Leakage in LLM Agent Skills: A Large-Scale Empirical Study. *arXiv:2604.03070* (2026).

[13] Zenghao Duan, Yuxin Tian, Zhiyi Yin, Liang Pang, Jingcheng Deng, Zihao Wei, Shicheng Xu, Yuyao Ge, and Xueqi Cheng. 2026. SkillAttack: Automated Red Teaming of Agent Skills through Attack Path Refinement. *arXiv:2604.04989* (2026).

[14] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2019. Diversity Is All You Need: Learning Skills without a Reward Function. In *International Conference on Learning Representations*.

[15] Lin Fan, Pengyu Dai, Zhipeng Deng, Haolin Wang, Xun Gong, Yefeng Zheng, and Yafei Ou. 2026. Evolving Medical Imaging Agents via Experience-driven Self-skill Discovery. *arXiv:2603.05860* (2026).

[16] Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. 2025. A Comprehensive Survey of Self-Evolving AI Agents: A New Paradigm Bridging Foundation Models and Lifelong Agentic Systems. *arXiv:2508.07407* (2025).

[17] Loris Gaven, Thomas Carta, Clement Romac, Cedric Colas, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. 2025. MAGELLAN: Metacognitive Predictions of Learning Progress Guide Autotelic LLM Agents in Large Goal Spaces. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*.

[18] Jingzhi Gong, Ruizhen Gu, Zhiwei Fei, Yazhuo Cao, Lukas Twist, Alina Geiger, Shuo Han, Dominik Sobania, Federica Sarro, and Jie M. Zhang. 2026. SkillMOO: Multi-Objective Optimization of Agent Skills for Software Engineering. *arXiv:2604.09297* (2026).

[19] Florian Holzbauer, David Schmidt, Gabriel Gegenhuber, Sebastian Schrittwieser, and Johanna Ullrich. 2026. Malicious Or Not: Adding Repository Context to Agent Skill Classification. *arXiv:2603.16572* (2026).

[20] Yingyong Hou, Xinyuan Lao, Huimei Wang, Qianyu Yao, Wei Chen, Bocheng Huang, Fei Sun, Yuxian Lv, Weiqi Lei, Xueqian Wen, Shengyang Xie, Pengfei Xia, and Zhujun Tan. 2026. MedSkillAudit: A Domain-Specific Audit Framework for Medical Research Agent Skills. *arXiv:2604.20441* (2026).

[21] Yinghan Hou and Zongyou Yang. 2026. SkillSieve: A Hierarchical Triage Framework for Detecting Malicious AI Agent Skills. *arXiv:2604.06550* (2026).

[22] Chenyi Huang, Haoting Zhang, Jingxu Xu, Zeyu Zheng, and Yunduan Lin. 2026. Bilevel Optimization of Agent Skills via Monte Carlo Tree Search. *arXiv:2604.15709* (2026).

[23] Ken Huang and Jerry Huang. 2025. Audited Skill-Graph Self-Improvement for Agentic LLMs via Verifiable Rewards, Experience Synthesis, and Continual Memory. *arXiv:2512.23760* (2025).

[24] Xu Huang, Junwu Chen, Yuxing Fei, Zhuohan Li, Philippe Schwaller, and Gerbrand Ceder. 2025. CASCADE: Cumulative Agentic Skill Creation through Autonomous Development and Evolution. *arXiv:2512.23880* (2025).

[25] Yi Huang, Bowen Zheng, Yunxi Dong, Hong Tang, Huan Zhao, S. M. Rakibul Hasan Shawon, and Hualiang Zhang. 2026. A Self-Evolving Agentic Framework for Metasurface Inverse Design. *arXiv:2604.01480* (2026).

[26] Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. 2026. XSkill: Continual Learning from Experience and Skills in Multimodal Agents. *arXiv:2603.12056* (2026).

[27] Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. 2026. SoK: Agentic Skills – Beyond Tool Use in LLM Agents. *arXiv:2602.20867* (2026).

[28] Yukun Jiang, Yage Zhang, Michael Backes, Xinyue Shen, and Yang Zhang. 2026. HarmfulSkillBench: How Do Harmful Skills Weaponize Your Agents? *arXiv:2604.15415* (2026).

[29] Zhengbo Jiao, Shaobo Wang, Zifan Zhang, Xuan Ren, Wei Wang, Bing Zhao, Hu Wei, and Linfeng Zhang. 2026. Agentic Proposing: Enhancing Large Language Model Reasoning via Compositional Skill Synthesis. *arXiv:2602.03279* (2026).

[30] Yeonsung Jung, Trilok Padhi, Sina Shaham, Dipika Khullar, Joonhyun Jeong, Ninareh Mehrabi, and Eunho Yang. 2025. Co-Evolving Agents: Learning from Failures as Hard Negatives. *arXiv:2511.22254* (2025).

[31] George Konidaris and Andrew G. Barto. 2009. Skill Discovery in Continuous Reinforcement Learning Domains Using Skill Chaining. In *Advances in Neural Information Processing Systems*, Vol. 22.

[32] Fangzhou Li, Pagkratios Tagkopoulos, and Ilias Tagkopoulos. 2025. SkillFlow: Scalable and Efficient Agent Skill Retrieval System. *arXiv:2504.06188* (2025).

[33] Guangrui Li, Yaochen Xie, Yi Liu, Ziwei Dong, Xingyuan Pan, Tianqi Zheng, Jason Choi, Michael Morais, Binit Jha, Shaunak Mishra, Bingrou Zhou, Chen Luo, Monica Cheng, and Dawn Song. 2026. The World Won't Stay Still: Programmable Evolution for Agent Benchmarks. *arXiv:2603.05910* (2026).

[34] Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026. Organizing, Orchestrating, and Benchmarking Agent Skills at Ecosystem Scale. *arXiv:2603.02176* (2026).

[35] Xiaoxiao Li. 2026. When Single-Agent with Skills Replace Multi-Agent Systems and When They Fail. *arXiv:2601.04748* (2026).

[36] Xiangyi Li, Yimin Liu, Wenbo Chen, Shenghan Zheng, et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv:2602.12670* (2026).

[37] Yu Li, Rui Miao, Zhengling Qi, and Tian Lan. 2026. ARISE: Agent Reasoning with Intrinsic Skill Evolution in Hierarchical Reinforcement Learning. *arXiv:2603.16060* (2026).

[38] Zhiyuan Li, Jingzheng Wu, Xiang Ling, Xing Cui, and Tianyue Luo. 2026. Towards Secure Agent Skills: Architecture, Threat Taxonomy, and Security Analysis. *arXiv:2604.02837* (2026).

[39] Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. 2026. SkillNet: Create, Evaluate, and Connect AI Skills. *arXiv:2603.04448* (2026).

[40] George Ling, Shanshan Zhong, and Richard Huang. 2026. Agent Skills: A Data-Driven Analysis of Claude Skills for Extending Large Language Model Functionality. *arXiv:2602.08004* (2026).

[41] Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. 2026. Graph of Skills: Dependency-Aware Structural Retrieval for Massive Agent Skills. *arXiv:2604.05333* (2026).

[42] Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. SimpleMem: Efficient Lifelong Memory for LLM Agents. *arXiv:2601.02553* (2026).

[43] Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. 2026. SkillForge: Forging Domain-Specific, Self-Evolving Agent Skills in Cloud Technical Support. *arXiv:2604.08618* (2026).

[44] Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. 2026. How Well Do Agentic Skills Work in the Wild: Benchmarking LLM Skill Usage in Realistic Settings. *arXiv:2604.04323* (2026).

[45] Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026. SKILL0: In-Context Agentic Reinforcement Learning for Skill Internalization. *arXiv:2604.02268* (2026).

[46] 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:2604.08377* (2026).

[47] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. 2018. Data-Efficient Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 31. 3307–3317.

[48] Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. 2026. Trace2Skill: Distill Trajectory-Local Lessons into Transferable Agent Skills. *arXiv:2603.25158* (2026).

[49] Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. 2026. AutoRefine: From Trajectories to Reusable Expertise for Continual LLM Agent Refinement. *arXiv:2601.22758* (2026).

[50] Yubin Qu, Yi Liu, Tongcheng Geng, Gelei Deng, Yuekang Li, Leo Zhang, Ying Zhang, and Lei Ma. 2026. Supply-Chain Poisoning Attacks Against LLM Coding Agent Skill Ecosystems. *arXiv:2604.03081* (2026).

[51] Sampriti Saha and Pranav Hemanth. 2026. Skilldex: A Package Manager and Registry for Agent Skill Packages with Hierarchical Scope-Based Distribution. *arXiv:2604.16911* (2026).

[52] Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. 2026. SkillFoundry: Building Self-Evolving Agent Skill Libraries from Heterogeneous Scientific Resources. *arXiv:2604.03964* (2026).

[53] Haochen Shi, Xingdi Yuan, and Bang Liu. 2026. Evolving Programmatic Skill Networks. *arXiv:2601.03509* (2026).

[54] Weijia Song, Jiashu Yue, and Zhe Pang. 2026. ABSTRAL: Automatic Design of Multi-Agent Systems Through Iterative Refinement and Topology Optimization. *arXiv:2603.22791* (2026).

[55] Yiqun Sun, Pengfei Wei, and Lawrence B. Hsieh. 2026. Don't Retrieve, Navigate: Distilling Enterprise Knowledge into Navigable Agent Skills for QA and RAG. *arXiv:2604.14572* (2026).

[56] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112, 1–2 (1999), 181–211.

[57] Zhen Tao, Riwei Lai, Chenyun Yu, Weixin Chen, Li Chen, Beibei Kong, Lei Cheng, Chengxiang Zhuo, Zang Li, and Qingqiang Sun. 2026. SAGER: Self-Evolving User Policy Skills for Recommendation Agent. *arXiv:2604.14972* (2026).

[58] Guiyao Tie, Jiawen Shi, Pan Zhou, and Lichao Sun. 2026. BadSkill: Backdoor Attacks on Agent Skills via Model-in-Skill Poisoning. *arXiv:2604.09378* (2026).

[59] Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. 2026. SkillX: Automatically Constructing Skill Knowledge Bases for Agents. *arXiv:2604.04804* (2026).

[60] 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:2305.16291* (2023).

[61] Jiayu Wang, Yifei Ming, Zixuan Ke, Shafiq Joty, Aws Albarghouthi, and Frederic Sala. 2026. SkillOrchestra: Learning to Route Agents via Skill Transfer. *arXiv:2602.19672* (2026).

[62] Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. 2025. Reinforcement Learning for Self-Improving Agent with Skill Library. *arXiv:2512.17102* (2025).

[63] Xudong Wang, Zebin Han, Zhiyu Liu, Gan Li, Jiahua Dong, Baichen Liu, Lianqing Liu, and Zhi Han. 2026. Lifelong Language-Conditioned Robotic Manipulation Learning. *arXiv:2603.05160* (2026).

[64] Xiaoxing Wang, Ning Liao, Shikun Wei, Chen Tang, and Feiyu Xiong. 2026. AutoAgent: Evolving Cognition and Elastic Memory Orchestration for Adaptive Agents. *arXiv:2603.09716* (2026).

[65] Zimu Wang, Yuling Shi, Mengfan Li, Zijun Liu, Jie M. Zhang, Chengcheng Wan, and Xiaodong Gu. 2026. EffiSkill: Agent Skill Based Automated Code Efficiency Optimization. *arXiv:2603.27850* (2026).

[66] Zihan Wang, Rui Zhang, Yu Liu, Chi Liu, Qingchuan Zhao, Hongwei Li, and Guowen Xu. 2026. Black-Box Skill Stealing Attack from Proprietary LLM Agents: An Empirical Study. *arXiv:2604.21829* (2026).

[67] Bowen Wei, Yunbei Zhang, Jinhao Pan, Kai Mei, Xiao Wang, Jihun Hamm, Ziwei Zhu, and Yingqiang Ge. 2026. ClawSafety: "Safe" LLMs, Unsafe Agents. *arXiv:2604.01438* (2026).

[68] Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. 2026. Group-Evolving Agents: Open-Ended Self-Improvement via Experience Sharing. *arXiv:2602.04837* (2026).

[69] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. 2025. EvolveR: Self-Evolving LLM Agents through an Experience-Driven Lifecycle. *arXiv:2510.16079* (2025).

[70] Xiyang Wu, Zongxia Li, Guangyao Shi, Alexander Duffy, Tyler Marques, Matthew Lyle Olson, Tianyi Zhou, and Dinesh Manocha. 2026. Co-Evolving LLM Decision and Skill Bank Agents for Long-Horizon Tasks. *arXiv:2604.20987* (2026).

[71] Zhe Wu, Donglin Mo, Hongjin Lu, Junliang Xing, Jianheng Liu, Yuheng Jing, Kai Li, Kun Shao, Jianye Hao, and Yuanchun Shi. 2026. K^2-Agent: Co-Evolving Know-What and Know-How for Hierarchical Mobile Device Control. *arXiv:2603.00676* (2026).

[72] Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. 2025. Live-SWE-agent: Can Software Engineering Agents Self-Evolve on the Fly? *arXiv:2511.13646* (2025).

[73] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. SkillRL: Evolving Agents via Recursive Skill-Augmented Reinforcement Learning. *arXiv:2602.08234* (2026).

[74] Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. MetaClaw: Just Talk – An Agent That Meta-Learns and Evolves in the Wild. *arXiv:2603.17187* (2026).

[75] Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. 2025. Agent0: Unleashing Self-Evolving Agents from Zero Data via Tool-Integrated Reasoning. *arXiv:2511.16043* (2025).

[76] Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. 2026. GraSP: Graph-Structured Skill Compositions for LLM Agents. *arXiv:2604.17870* (2026).

[77] Renjun Xu and Yang Yan. 2026. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. *arXiv:2602.12430* (2026).

[78] Cheng Yang, Xuemeng Yang, Licheng Wen, Daocheng Fu, Jianbiao Mei, Rong Wu, Pinlong Cai, Yufan Shen, Nianchen Deng, Botian Shi, Yu Qiao, and Haifeng Li. 2025. Learning on the Job: An Experience-Driven Self-Evolving Agent for Long-Horizon Tasks. *arXiv:2510.08002* (2025).

[79] Yongjin Yang, Sinjae Kang, Juyong Lee, Dongjun Lee, Se-Young Yun, and Kimin Lee. 2025. Automated Skill Discovery for Language Agents through Exploration and Iterative Feedback. *arXiv:2506.04287* (2025).

[80] Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. 2026. AutoSkill: Experience-Driven Lifelong Learning via Skill Self-Evolution. *arXiv:2603.01145* (2026).

[81] Renos Zabounidis, Yue Wu, Simon Stepputtis, Woojun Kim, Yuanzhi Li, Tom Mitchell, and Katia Sycara. 2026. SCALAR: Learning and Composing Skills through LLM Guided Symbolic Planning and Deep RL Grounding. *arXiv:2603.09036* (2026).

[82] Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, Zhaoyang Liu, Bolin Ding, and Jingren Zhou. 2025. AgentEvolver: Towards Efficient Self-Evolving Agent System. *arXiv:2511.10395* (2025).

[83] Aimin Zhang, Jiajing Guo, Fuwei Jia, Chen Lv, Boyu Wang, and Fangzheng Li. 2026. EvoAgent: An Evolvable Agent Framework with Skill Learning and Multi-Agent Delegation. *arXiv:2604.20133* (2026).

[84] Di Zhang. 2026. AgentDevel: Reframing Self-Evolving LLM Agents as Release Engineering. *arXiv:2601.04620* (2026).

[85] Dengjia Zhang, Xiaoou Liu, Lu Cheng, Yaqing Wang, Kenton Murray, and Hua Wei. 2026. SELAUR: Self Evolving LLM Agent via Uncertainty-aware Rewards. *arXiv:2602.21158* (2026).

[86] 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. 2026. Co-EvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification. *arXiv:2604.01687* (2026).

[87] Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. 2026. MemSkill: Learning and Evolving Memory Skills for Self-Evolving Agents. *arXiv:2602.02474* (2026).

[88] Xiaoying Zhang, Zichen Liu, Yipeng Zhang, Xia Hu, and Wenqi Shao. 2026. RetroAgent: From Solving to Evolving via Retrospective Dual Intrinsic Feedback. *arXiv:2603.08561* (2026).

[89] Xing Zhang, Guanghui Wang, Yanwei Cui, Wei Qiu, Ziyuan Li, Bing Zhu, and Peiyang He. 2026. Experience Compression Spectrum: Unifying Memory, Skills, and Rules in LLM Agents. *arXiv:2604.15877* (2026).

[90] Zhang Zhang, Shuqi Lu, Hongjin Qian, Di He, and Zheng Liu. 2026. AgentFactory: A Self-Evolving Framework Through Executable Subagent Accumulation and Reuse. *arXiv:2603.18000* (2026).

[91] Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qisheng Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. 2026. SkillFlow: Benchmarking Lifelong Skill Discovery and Evolution for Autonomous Agents. *arXiv:2604.17308* (2026).

[92] Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. 2025. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. *arXiv:2504.07079* (2025).

[93] Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, Zhong-Zhi Li, Yingying Zhang, Le Song, and Qianli Ma. 2025. LifelongAgentBench: Evaluating LLM Agents as Lifelong Learners. *arXiv:2505.11942* (2025).

[94] Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. 2025. Lifelong Learning of Large Language Model based Agents: A Roadmap. *arXiv:2501.07278* (2025).

[95] YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. 2026. SkillRouter: Skill Routing for LLM Agents at Scale. *arXiv:2603.22455* (2026).

[96] Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. 2026. SkillLearnBench: Benchmarking Continual Learning Methods for Agent Skill Generation on Real-World Tasks. *arXiv:2604.20087* (2026).

[97] Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. 2026. Memento-Skills: Let Agents Design Agents. *arXiv:2603.18743* (2026).

A Search and Screening Procedure

The corpus was assembled by iterative database search and backward/forward snowballing rather than by a single PRISMA-style query. We searched arXiv, Semantic Scholar, Google Scholar, OpenReview, and venue proceedings using combinations of *LLM agent skill*, *agentic skill*, *skill library*, *self-evolving agent*, *lifelong agent*, *skill routing*, *SKILL.md*, *agent skill safety*, *skill benchmark*, and named-system queries discovered during snowballing. Candidates were screened first by title/abstract and then by PDF when the abstract suggested a persistent artifact, skill-like package, evolving library, or skill-specific safety/evaluation surface. We excluded papers whose only adaptation object was model weights, prompt optimization, generic tool use, or episodic memory without an invocable interface, unless the paper directly evaluated how such artifacts become reusable skills. The final workshop audit set contains 94 papers; the representative master sheet in Appendix D codes 82 systems where the mechanism is concrete enough to assign all fields.

Inclusion criteria. Papers from 2023–2026 satisfying at least one of: (i) the paper proposes a mechanism that adds, edits, prunes, distills, routes, transfers, or composes an agent skill artifact; (ii) the paper provides infrastructure or a benchmark that changes how skill libraries are stored, retrieved, evaluated, or governed; or (iii) the paper documents a safety or deployment failure mode specific to skill artifacts.

Boundary works. Treated as context rather than corpus members: SoK-Skills [27]; Xu and Yan [77]'s 2026 position paper; MAGELLAN's autotelic curriculum [17]; the Experience Compression Spectrum framework [89]; and adjacent surveys on self-evolving agents [16] and lifelong LLM agents [94]. Classical options and HRL anchors [4, 14, 31, 47, 56] are cited but not counted in the 94-paper corpus total. The April 24, 2026 cutoff is an audit boundary, not a claim that the field has stabilized.

Evidence grades. We grade evidence qualitatively. **A**: multiple controlled ablations or one clean ablation plus independent corroboration. **B**: one controlled study or a strong benchmark/deployment measurement. **C**: convergent benchmark behavior without a clean causal ablation. **D**: architectural corroboration only. We avoid cross-paper leaderboards unless the harness is shared, because backbone, tool surface, context budget, and evaluator often change together.

Note schema and conflict resolution. Each paper was read into a common note schema: problem framing, mechanism, control action, experimental setup, headline results, ablations, limitations, closest peers, survey takeaway. When duplicate notes disagreed, conflicts were resolved against the PDF. The audit was not independently double-coded and therefore does not report inter-coder reliability; we instead expose the coding sheet, evidence grades, and aggregate summaries below so that disagreements can be traced to specific rows.

B The Six Senses of "Skill"

The terminology distinguishes artifact types; the formalism extends them with edit, verification, and lineage. Five dynamic properties largely determine which triggers, operators, and signals a method can support: whether the artifact is *executable*, *editable in place*, *portable across models*, *inspectable by humans*, and attached to a *verification handle*.

Table 5. Six senses of "skill" across the surveyed methods. Y= fully satisfies; ∼ = partial / indirect; – = does not satisfy.

Method	Sense	Exec.	Edit	Port.	Insp.	Verif. handle
Voyager [60]	Code	Y	Y	∼	Y	unit test
LATM [7]	Code	Y	Y	∼	Y	unit test
AgentFactory [90]	Code	Y	Y	∼	Y	meta-agent
LIVE-SWE [72]	Code	Y	Y	∼	Y	rollout
SAGE [62]	Code	Y	Y	∼	Y	env. reward
ERL [2]	NL lesson	–	Y	Y	Y	indirect
RetroAgent [88]	NL lesson	–	Y	Y	Y	indirect
EvolveR [69]	NL lesson	–	Y	Y	Y	indirect
MetaClaw [74]	SKILL.md	Y*	Y	Y	Y	L3 code gate
Memento [97]	SKILL.md+case	Y*	Y	Y	Y	L3 code gate
Trace2Skill [48]	SKILL.md	Y*	Y	Y	Y	rubric judge
AutoRefine [49]	SKILL.md	Y*	Y	Y	Y	rubric judge
SkillFlow-Bench [91]	SKILL.md patch	Y*	Y	Y	Y	benchmark verifier
ABSTRAL [54]	SKILL.md/doc	–	Y	Y	Y	trace evidence
SkillsCrafter [63]	Parametric	Y†	–	–	–	behavioral probe
SKILL0 [45]	Parametric	Y†	–	–	–	behavioral probe
SELAUR [85]	Parametric	Y†	–	–	–	behavioral probe
LSE [11]	Parametric	Y†	∼	–	∼	behavioral probe
SimpleMem [42]	Memory/traj.	–	∼	Y	Y	indirect
MUSE [78]	Memory/traj.	–	∼	Y	Y	indirect
CASCADE [24]	Memory/traj.	–	∼	Y	Y	indirect
XSkill [26]	Skill+exp.	∼	Y	Y	Y	indirect
SkillFlow-2025 [32]	Registry-retrieved skill	∼	–	Y	Y	retrieval eval
Co-Evolving [30]	Cap. label	–	∼	Y	Y	none
GEA [68]	Cap. label	–	∼	Y	Y	none

*Executable when L3 attaches code or MCP resources. †Executable through the compatible base model.

C Worked Instantiation: AutoRefine as a Library Transition

The 7-tuple is intended to describe concrete system behavior. Consider an AutoRefine-style maintenance cycle [49]. A skill document in the current library can be written as

$$\mathcal{S}_t^{(i)} = \langle C_t, \pi_t, T_t, R_t, \varphi_t, \nu_t, \prec_t \rangle,$$

where C_t is the task or state description under which the skill should be retrieved, π_t is the SKILL.md instruction body plus optional executable helper, T_t is the success/failure or budget condition observed during use, and R_t is the call signature or natural-language invocation handle.

One step. After a trajectory exposes a failure or redundancy, the trigger is τ_t = TaskEnd or Periodic, and the signal r_t is a critique, execution trace, or downstream utility measurement. The update rule chooses an operator vector such as

$$\vec{u}_t = \{(\textsc{Refine}, \text{patch ambiguous step}),\\ (\textsc{Merge}, \text{combine duplicate routines}),\\ (\textsc{Prune}, \text{quarantine low-utility skill})\}.$$

Each realized edit yields a candidate $\mathcal{S}^*$. The admission gate evaluates $v_t(\mathcal{S}^*)$ using the method's judge, execution feedback, or consistency checks. If the candidate passes, $\mathcal{S}^*$ enters $\mathcal{L}_{t+1}$ and $\prec_{t+1}$ records that it supersedes or merges earlier artifacts; if it fails, $\mathcal{L}_{t+1}$ retains the prior version or marks the candidate for later review. The transition is therefore not a single append: it is a gated composition of Refine, Merge, and Prune over a versioned artifact store.

What the lens makes auditable. Every claim is exposable: how many candidates φ_t proposed, how many v_t rejected, how often admitted skills were retrieved, which failures triggered maintenance, and whether the next library state $\mathcal{L}_{t+1}$ improved future tasks. The same decomposition applies to NL procedures (verifier → replay or judges), workflow graphs (→ dependency consistency), and skill-aware RL (→ reward/verifier quality).

D Master Coding Sheet

Table 6 instantiates the seven coding fields of §4 for representative systems. Cells use these abbreviations: *Artifact*: Code, NL, MD = SKILL.md, LoRA, Mix; *Clock*: fast (in-context), slow (parametric), 2TS (two-timescale); *Trigger*: Task, Fail, Per (periodic), User, RL; *Operators*: A=Add, R=Refine, M=Merge, S=Split, P=Prune, D=Distill, B=Abstract, C=Compose, W=Rewrite, K=Rerank; *Signal*: Rew (env reward), Exec (execution feedback), Crit (NL critique), Judge, XUser (cross-user aggregation), Teach; *Storage*: flat, tree, DAG, graph, subsp (subspace), ontol (ontology).

Table 6. Master coding table for representative dynamic-skill systems across the seven coding fields. The table is a coding sheet that supports the lifecycle-family taxonomy of Table 1.

Method	Cluster	Artif.	Clock	Trig.	Operators	Signal	Store	Headline
Foundational								
Voyager [60]	Found.	Code	fast	Task	A	Rew+Exec	flat	3.3× items; 15.3× tech-tree
LATM [7]	Found.	Code	fast	Task	A	Exec	flat	Tool-maker beats few-shot
Skill-aware RL								
SAGE [62]	RL	Code	2TS	Task	A,R	Rew	flat	Skill-integrated RL on App-World
SkillRL [73]	RL	Code	slow	RL	A,D	Rew	tree	Hier. skill-conditioned RL
AgentEvolver [82]	RL	Mix	2TS	RL	A,R,D	Rew+Crit	flat	Self-question / navigate / attribute
CODE-SHARP [6]	RL	Code	2TS	Fail	A,R,M,W	Rew+Judge	DAG	4 mutations + prereq DAG
Agentic-Prop. [29]	RL	Mix	slow	RL	A,R,P,D,B,C	Teach+Judge	DAG	91.6% AIME-25 (30B, 11K traj.)
COS-PLAY [70]	RL	Mix	2TS	RL	A,R,P,D	Rew	flat	Co-evolves decision + skill bank
Heuristic / lesson memory								
ERL [2]	Lesson	NL	fast	Task	A,R	Crit	flat	Task-end retrospective lessons
RetroAgent [88]	Lesson	NL	fast	Fail	A,R	Crit+Judge	flat	Dual intrinsic feedback
MUSE [78]	Lesson	NL	fast	Per	A,R,D	Crit	tree	Long-horizon productivity memory
EvolveR [69]	Lesson	NL	fast	Task	A,R,W	Crit	flat	Strategic-principle evolution
Memento [97]	Lesson	MD	fast	Task	A,R,K	Crit+Judge	tree	Case memory + SKILL.md rerank
XSkill [26]	Lesson	Mix	fast	Task	A,R,M,P,B,K	Crit+Judge	flat	Visual dual skill/exp. store
SAGER [57]	Lesson	NL	fast	User	A,R,W	XUser	flat	Per-user policy skills for rec.
Executable skill libraries								
SkillWeaver [92]	ExecLib	Code	fast	Task	A,R,P	Judge+Exec	flat	Propose-practice-synth.-hone
AgentFactory [90]	ExecLib	Code	fast	User	A,R,C	Judge	DAG	Subagents as evolvable skills
EvoSkill [3]	ExecLib	Code	fast	Fail	A,R,P	Judge+Exec	flat	Failure-driven + Pareto admission
CoEvoSkills [86]	ExecLib	MD	fast	Task	A,R,B	Judge	flat	Co-evolving verifier loop
PSN [53]	ExecLib	Code	fast	Fail	A,R,M,W,K,P	Crit+Judge	graph	5 refactors + symbolic credit
SkillCraft [9]	ExecLib	Code	fast	Task	A,C,R	Exec+Judge	DAG	Verified compositional MCP
LIVE-SWE [72]	ExecLib	Code	fast	Task	A,R	Exec	flat	Runtime scaffold synthesis
CASCADE [24]	Lesson	Mix	fast	Per	A,D,P	Teach	tree	Scientific skill consolidation
SkillX [59]	ExecLib	Code	fast	Per	A,S,B,K	Judge	tree	Automatic hierarchical library

continued on next page

Table 6 (continued)

Method	Cluster	Artif.	Clock	Trig.	Operators	Signal	Store	Headline
Trace2Skill [48]	ExecLib	MD	fast	Task	A,M,D	Judge	flat	Prevalence-weighted consolidation
SkillClaw [46]	ExecLib	MD	fast	User	A,R,M,K	XUser	flat	Cross-user skill evolution
AutoSkill [80]	ExecLib	MD	fast	User	A,R	XUser	flat	Training-free personalized bank
AutoRefine [49]	ExecLib	MD	fast	Per	A,R,M	Crit+Judge	flat	Dual-form skills + maintenance
MemSkill [87]	ExecLib	MD	fast	Task	A,R,K	Crit	tree	Meta-memory skill banks
Wild-Skills [44]	ExecLib	Code	fast	Per	A,P,K	Rew+Judge	flat	Utility under realistic retrieval
CUA-Skill [10]	ExecLib	Code	fast	Task	A,B	Exec	flat	Parameterized GUI skills
ABSTRAL [54]	Infra	MD	fast	Task	A,R,C	Crit	graph	Multi-agent design via SKILL.md
SkillFoundry [52]	ExecLib	Mix	fast	Per	A,R,M,P,B	Exec+Judge	tree	Scientific contracts + provenance
SkillForge [43]	ExecLib	MD	fast	Fail	A,R,W	Judge	flat	Failure diagnosis in cloud support
SkillMOO [18]	ExecLib	MD	fast	Fail	R,P,K	Judge+Exec	flat	Pareto pass/cost optimization
Bilevel-MCTS [22]	ExecLib	MD	fast	Per	R,W	Judge	flat	MCTS over package structure
Metasurface [25]	ExecLib	Mix	fast	Fail	A,R	Exec	flat	Physics-verified scientific skills
EvoAgent [83]	ExecLib	Mix	fast	User	A,R,K,C	XUser+Judge	tree	Multi-file skills + delegation
MACRO [15]	ExecLib	Code	2TS	Task	A,C,D	Rew+Exec	flat	Discovers composite medical tools
Parametric / training-time								
SELAUR [85]	Param	LoRA	slow	Fail	D	Rew(unc)	subsp	Uncertainty-reshaped fail-rewards
SKILL0 [45]	Param	LoRA	slow	Per	D,B	Teach	subsp	Helpfulness-decaying curricula
LSE [11]	Param	Mix	slow	Task	D,R	Rew	subsp	Trained prompt-context evolution
SkillsCrafter [63]	Param	LoRA	slow	Per	D,M,K	Rew	subsp	LoRA bank + semantic subspace
K2-Agent [71]	Param	Mix	2TS	Task	A,R,D	Rew+Teach	tree	Decl.-proc. co-evolution
MetaClaw [74]	Param	Mix	2TS	Per	A,R,D	Rew	flat	Fast-skill / slow-weight adapt.
Co-Evolving [30]	Param	Mix	slow	Fail	A,D,P	Rew+Crit	flat	Hard-negative failure training
Agent0 [75]	Param	Mix	2TS	RL	A,D	Rew	flat	Zero-data curric.-executor co-ev
Tool-R0 [1]	Param	Mix	slow	RL	A,D	Rew	flat	Dual self-play for tool-use
SCALAR [81]	Param	Mix	slow	RL	D,B	Teach	flat	Co-adapt difficulty + env
ARISE [37]	Param	LoRA	slow	RL	D,K	Rew	subsp	Swarm PPO + PSO actions
EXIF [79]	Param	LoRA	slow	RL	D,B	Rew+Teach	flat	Exploration-first skill data
Infrastructure & governance								
AgentSkillOS [34]	Infra	MD	fast	Per	A,C,S,P,K	Judge	tree	Capability tree + DAG orch.
SkillRouter [95]	Infra	Code	fast	Per	K	Judge	flat	Full-text retrieval 80K skills
SkVM [8]	Infra	Code	fast	User	A,R,C	Exec	DAG	Skills as compilable code
SkillNet [39]	Infra	MD	fast	Per	A,K,P	Judge	graph	Ontology + relation graph
SkillOrchestra [61]	Infra	MD	fast	Per	K,C	Judge	ontol	Skill handbooks for routing
SkillFlow-2025 [32]	Infra	MD	fast	Task	K	Judge	flat	Multi-stage community-skill retrieval
SkillFlow-Bench [91]	Eval	MD	fast	Task	A,R,P	Crit+Judge	flat	166 tasks / 20 task families
AutoAgent [64]	Infra	Mix	2TS	Task	A,R,C	Crit+Judge	DAG	Evolves tools, peers, and self
AgentDevel [84]	Infra	MD	fast	User	A,R,K,P	XUser	graph	Release engineering + lineage
GEA [68]	Infra	Mix	2TS	Per	A,R,M	Judge+Rew	ontol	Group-based framework evolution
Single-Ag.Sk. [35]	Infra	Code	fast	Per	A,M,D,P	Judge	flat	Multi→single compilation
EffiSkill [65]	Infra	Code	fast	Per	A,B,K	Rew	flat	Mined operator skills
Graph of Skills [41]	Infra	Code	fast	User	K,C	Rew	graph	Dependency-aware retrieval
GraSP [76]	Infra	Code	fast	Task	C,R	Exec	DAG	Typed DAG composition + repair
Skilldex [51]	Infra	MD	fast	User	A,K,P	Judge	tree	Package manager + scoped registry
Corpus2Skill [55]	Infra	MD	fast	User	A,S,K	Judge	tree	Navigable enterprise QA skills
SkillRepoMining [5]	Infra	MD	fast	User	A,B	Judge	flat	GitHub repo mining to SKILL.md
AgentSkills-Data [40]	Eval	MD	fast	Per	K,P	Judge	—	40K+ public-skill ecosystem study
SkillLearnBench [96]	Eval	MD	fast	Task	A,R	Judge	flat	Continual skill-generation benchmark
Safety & audit								
ASG-SI [23]	Safety	Mix	2TS	Task	A,R,P,D	Rew+Judge	graph	Audited skill graph + verif. rewards

continued on next page

Table 6 (continued)

Method	Cluster	Artif.	Clock	Trig.	Operators	Signal	Store	Headline
ClawSafety [67]	Safety	MD	fast	User	P	Judge	—	Highest-ASR vector in personal agents
Secure-Skills [38]	Safety	MD	fast	User	P	Judge	—	Lifecycle threat taxonomy
Supply-Chain [50]	Safety	MD	fast	User	P	Judge	—	DDIPE poisoning attacks
SkillSieve [21]	Safety	MD	fast	User	P	Judge	—	Hierarchical malicious-skill triage
SkillAttack [13]	Safety	MD	fast	User	P	Judge	—	Automated attack-path red teaming
BadSkill [58]	Safety	Mix	fast	User	P	Exec	—	Model-in-skill backdoors
CredLeak [12]	Safety	Mix	fast	User	P	Exec+Judge	—	Cross-modal secret leakage
HarmfulSkillBench [28]	Safety	MD	fast	User	P	Judge	—	Harmful-but-valid skills
SkillStealing [66]	Safety	MD	fast	User	K	Judge	—	Black-box skill extraction
Malicious-or-Not [19]	Safety	MD	fast	User	P	Judge	—	Repository-aware classification
MedSkillAudit [20]	Safety	MD	fast	User	P,R	Judge	—	Medical release-readiness audit

E Aggregate Patterns in the Coding Sheet

The master coding sheet is a representative coding artifact, not a statistical sample. Still, aggregate counts make the taxonomy less anecdotal. Table 7 summarizes the most relevant distributions over the 82 fully coded representative systems.

Table 7. Aggregate patterns from the representative coding sheet. Counts are row-level operator mentions, so one system can contribute multiple operators.

Coding aggregate	Observed distribution	Interpretation
Operator frequency	A:56, R:42, P:29, K:23, D:22, C:12, B:11, M:10, W:6, S:3	Growth and local refinement dominate; negative and structural operators are rarer, matching the monotonic-growth concern.
Family coverage	ExecLib:22, Infra:17, Param:12, Safety:12, Lesson:8, RL:6, Eval:3, Foundational:2	The corpus is no longer just executable libraries; infrastructure and safety are large enough to shape the taxonomy.
Flat-store operator mix	Flat stores account for 32 A and 22 R mentions but only 9 P and 5 M mentions	Flat storage makes addition/refinement cheap but gives weaker built-in support for pruning and merging.
Structured-store operator mix	Tree/DAG/graph stores concentrate S/C/K/P relative to flat stores	Structure buys compositionality and maintenance handles, but raises rollback and dependency-management cost.
Safety rows	Safety/audit rows mostly code P/K/R rather than A/D/C	Safety systems focus on filtering, quarantine, and release readiness rather than proposing new capabilities.

F Lifecycle Architecture: Per-Stage Detail

Table 8. Lifecycle architecture for dynamic skill systems. Each stage carries a different design question, set of operators, evidence sources, and characteristic failure mode.

Stage	Design question / evidence	Operators	Failure
Evidence acquisition	What observation justifies a library change (trajectories, rubrics, visual rollouts, rewards, failure tickets, external resources)	—	noisy evidence, non-stationary utility
Proposal / externalization	How is evidence converted into an artifact (retrospective lessons, executable proposals, file patches, repository mining)	Add, Abstract	task-specific or overgeneral skills
Verification & admission	What gate decides whether the candidate enters (execution checks, Pareto admission, surrogate tests, multi-test validation, rollback validation, learned gates)	Refine, Rewrite	plausible but wrong skills; verifier hacking
Organization & storage	Where does the admitted artifact live (capability trees, prerequisite DAGs, invocation graphs, ontology graphs, dependency retrieval, scoped packages)	Split, Merge, Rerank	flat-retrieval collapse, stale structure
Retrieval & composition	Which skills affect the next action (80K routing, realistic distractors, dependency-aware retrieval, typed DAG composition, visual adaptation)	Rerank, Compose	usage without utility, distractor load
Maintenance & repair	How is the library kept compact and correct (pruning/merging, dual-store consolidation, audits, maturity gating, failure diagnosis, inflation)	Prune, Merge, Refine, Rewrite	unbounded growth, negative transfer
Distillation & portability	What moves between artifacts, weights, agents (parametric distillation, peer transfer)	Distill, Abstract	parametric collapse, compatibility failure
Governance & provenance	Can edits be audited, attributed, and rolled back (audited graph, release workflow, threat taxonomy, scanners, registry studies, release audit)	logged Add–Prune sequence	skill injection, leakage, attribution loss

G Verification Architectures

Table 9. Verification architectures across the surveyed methods. Timing codes: W = write-time, I = invocation-time, M = maintenance-time.

Form	Time	Admission	Cost	Cov.	Methods
Execution tests / rollouts	W	hard	high	narrow	Voyager, LATM, SkillWeaver, EvoSkill, LIVE-SWE, SkillCraft, SkillFoundry, Metasurface, MACRO
Judge-LLM / meta-agent	W+M	hard / Pareto	medium	medium	Trace2Skill, CoEvoSkills, AutoRefine, AgentSkillOS, SkillNet, SkillOrchestra, CODE-SHARP, AgentFactory, SkillForge, MedSkillAudit
Ensemble voting	W+M	maturity-gated	medium	wide	Agentic-Proposing
Symbolic / audited	W	graph integrity	medium	wide	ASG-SI
Rollback validation	W+M	Δ-success $\leq$ 20%	medium	behavioral	PSN
Economic / utility	I+eviction	utility threshold	low	deferred	Wild-Skills, EffiSkill
Security triage / red-team	W+I	quarantine / review	medium	adversarial	SkillSieve, ClawSafety, SkillAttack, Supply-Chain-Poisoning, BadSkill, Credential Leakage

Execution gates are precise but narrow: passing one contract does not prove broad transfer. Judge gates are broader but vulnerable to evaluator drift and rubric hacking. Rollback gates directly measure behavioral regression on the probe set. Utility gates are cheap and deployment-realistic but may admit harmful skills before evidence accumulates. Higher-capacity systems use staged admission: tentative, quarantined, promoted, demoted, or rolled back rather than simply accepted or rejected.

H Two-Timescale Decoupling Modes

Table 10. Two-timescale decoupling modes. SA = shared-artifact (fast and slow loops edit and distill the same textual skill); **SS** = shared-signal; **SC** = shared-curriculum; **SV** = shared-verifier.

Method	Mode	Fast	Slow	Promoted on	Slow trigger
K2-Agent [71]	SA	SKILL.md+case	LoRA/adapter	reward ∩ prevalence	task end
MetaClaw [74]	SA	SKILL.md	LoRA	prevalence	periodic
AgentEvolver [82]	SS	NL heuristic	policy weights	RL replay buffer	RL update
Tool-R0 [1]	SS	tool proposals	policy weights	dual self-play reward	RL step
SAGE [62]	SS	Python skill fn	policy weights	skill-integrated reward	task end
COS-PLAY [70]	SS	skill-bank	GRPO adapters	rollout reward + utility	co-evolution
Agent0 [75]	SC	code proposals	executor weights	coverage / curriculum	co-evolution
SCALAR [81]	SC	env / curation	policy weights	teacher signal	RL step
ASG-SI [23]	SV	audited graph	policy weights	audit-gated reward	RL step
Memento [97]	—	SKILL.md+case	(no slow loop)	—	—
LSE [11]	—	prompt context	prompt+weight	reward	—
Co-Evolving [30]	—	code+critique	policy weights	hard-negative pool	batch end
Agentic-Proposing [29]	—	skill proposals	policy weights	ensemble pass	RL step

Two-timescale adaptation is not automatically superior. A noisy fast-loop library gives the slow loop noisy targets, so distillation can compress mistakes as well as discoveries. An over-conservative fast loop leaves useful knowledge external and expensive. The open variable is the consolidation schedule: when to distill, what to distill, and whether the slow-loop result should replace, augment, or merely rerank the external library.

I Lifecycle Benchmarks

Table 11. Benchmark coverage over dynamic-skill lifecycle dimensions.

Benchmark	Primary lifecycle coverage	Self-gen.	Revision	Trajectory	Use → utility
SkillsBench	skill usefulness	Y	—	—	—
Wild-Skills	realistic retrieval, 34K-pool distractors	—	—	~	Y
SkillRouter	full-text routing at 80K scale	—	—	—	~
Graph of Skills	dependency-aware retrieval 200–2K skills	—	—	—	Y
LifelongAgentBench [93]	lifelong sequences	~	~	Y	—
ProEvolve [33]	programmable distribution shift	—	—	Y	—
SkillFlow-Bench	166 tasks, 20 families; patches, repair, reuse	Y	Y	Y	Y
SkillLearnBench	continual skill generation, 20 verified tasks	Y	Y	Y	Y
ClawSafety	skill-injection attack benchmark	—	—	—	Y
HarmfulSkillBench	harmful-skill prevalence and invocation	—	—	—	Y
MedSkillAudit	domain release-readiness audit, 75 skills	—	Y	—	~

A trajectory-aware protocol should report the library as a time series, with four elements: (i) performance, skill count, and retrieval quality at a grid of task indices or library sizes; (ii) operator velocities for Add, Refine, Merge, Prune; (iii) a drift schedule or family-transfer condition; (iv) a maintenance-off or repair-off ablation.

J Infrastructure Dimensions

Infrastructure choices predetermine which operator vocabulary is economical. The fast/costly asymmetry below is the core point: storage topology and pipeline architecture decide which operators run at deployment velocity, which is why the operator–storage diagonal of Table 6 is sparsely populated.

Table 12. Four infrastructure dimensions and their operator economy.

Dim.	Class	Reps.	Fast ops	Costly ops
Pkg.	SKILL.md	SkillClaw, AutoSkill, MetaClaw, Skilldex	A,R	C
	Tool schema	MCP/plugin manifests	A,C	B
	Skill+memory	XSkill, Memento, SAGER	A,K	M,P
	Compiled corpus	Corpus2Skill, SkillRepoMining, SkillFoundry	A,B	P
	Trajectory	SimpleMem, MUSE	A,K	B,D
Storage	Flat embedding	EvoSkill, SkillWeaver, Agentic-Prop.	A	M
	Hierarchical	AgentSkillOS, SkillX, Corpus2Skill	S	P
	Graph / dependency	PSN, CODE-SHARP, Graph of Skills, GraSP	C	P
	Typed ontology	SkillOrchestra	C(typed)	R
Market	Pull-model	Skilldex	A,K	M
	Push-model	SkillClaw cross-user	A	M,P
	Hybrid	AutoSkill dual review	A,K	M,P
	Security scanner	SkillSieve, Mal-or-Not, CredLeak	P	R
Pipeline	Inline edit	PSN, CODE-SHARP	R,W	P
	Log-and-apply	AutoRefine, AgentSkillOS	P	R
	Shadow exec.	MetaClaw, SKILL0	D	R
	Release audit	MedSkillAudit, AgentDevel	P	R

K Eight Safety Surfaces in Detail

Li et al. [38] organize threats across creation, distribution, deployment, and execution; we map that phase structure onto the operator-and-artifact-store formalism.

1. Prompt injection through admitted skills. ClawSafety [67] reports malicious skill files as the highest-ASR injection vector (69.4% averaged across backbones). SkillAttack [13] shows the complementary red-team view: benign-looking skills can become exploitable when an attacker refines prompts into attack paths.

2. Supply-chain poisoning at publication and installation. Supply-Chain-Poisoning [50] constructs DDIPE attacks over 1,070 adversarial skills across 15 MITRE ATT&CK categories, with bypass rates from 11.6% to 33.5% depending on defense, and 2.5% evading both static and alignment filters. BadSkill [58] bundles backdoored models with 97.5–99.5% ASR while preserving benign accuracy.

3. Harmful but well-formed skills. HarmfulSkillBench [28] finds 4,858 harmful skills among 98,440 collected and shows installed harmful skills can raise harm scores under implicit invocation. This is a policy-governance problem rather than malware detection.

4. Credential and secret leakage. Credential Leakage [12] samples 17,022 skills from a 170,226-skill SkillsMP population, identifies 520 affected skills and 1,708 issues, and reports 76.3% of cases require joint NL+code analysis. Debug logging accounts for 73.5% of issues because stdout is fed back to the LLM.

5. Admission scanner limits and false positives. SkillSieve [21] filters 86% of skills at zero API cost and reaches 0.800 F1 at $0.006 per skill. Malicious-or-Not [19] shows scanner-only classification overstates risk; adding repository context reduces 2,887 scanner-flagged combinations to 0.52% remaining as flagged. Admission gates need both content and provenance/context analysis.

6. Ownership, confidentiality, and skill theft. SkillStealing [66] demonstrates black-box extraction of paid or proprietary skill behavior with few interactions, with substantial semantic leakage even when exact text recovery is low. The victim may be a skill author or registry operator rather than the end user.

Table 14. Eight open problems for the dynamic-skills agenda, each paired with a concrete first experiment and an evidence-basis tag.

Open problem (basis)	Concrete first experiment	Likely obstacle
O1. Compositional verifiers (*extrapolated*)	Replace one CODE-SHARP mutation with a PSN-style rollback-gated variant; measure velocity-vs-admission frontier.	Different graph semantics (invocation vs prerequisite).
O2. Admission under non-stationary distributions (*extrapolated*)	Wrap a skill-aware RL verifier in a utility tracker that shifts the admission threshold; test on a planted distribution-shift benchmark.	No such benchmark currently exists.
O3. Theoretically grounded maintenance schedules (*measured need; extrapolated theory*)	Derive optimal maintenance frequency f^* from a cost model; compare to periodic-distillation defaults in MetaClaw, SKILL0.	Bounds depend on unobserved drift rate.
O4. Parametric-collapse prevention (*speculative*)	Run any two-timescale system for 10× its reported distillation windows; track proposal diversity.	Compute budget for long horizons.
O5. Retrieval scaling beyond the knee (*measured at moderate; extrapolated to marketplace*)	Evaluate learned routing, graph retrieval, skill compilation, and hybrid parametric–retrieval at 10^4–10^6 skills.	Apples-to-apples comparison across storage structures.
O6. Cross-library skill portability (*measured + extrapolated*)	Extend CoEvoSkills / XSkill transfer into a controlled cross-product of author backbone, receiving backbone, harness, context budget, tool API.	Tool/tokenizer/context mismatches; transfer can raise Pass@4 while lowering Avg@4.
O7. Governance, provenance, and attribution (*measured attacks + partial defenses*)	Scale ASG-SI-style provenance + SkillSieve-style triage; log-based logical undo vs cryptographic provenance vs snapshots at 10^3 edits/day.	Provenance and scanner overhead under high operator velocity.
O8. Honest dynamic benchmarks (*measured benchmark gaps*)	Combine SkillFlow-Bench sequential patches with Wild-Skills retrieval realism; report global library trajectory, operator velocity, usage-vs-utility.	Standardizing a shared global-library protocol across task families.

7. Domain-specific release readiness. MedSkillAudit [20] evaluates 75 medical research skills, finds 57.3% fall below the Limited Release threshold, and reports automated-audit $\mathrm{ICC}(2,1) = 0.449$ against expert consensus.

8. Attribution loss and composition misuse. Maintenance operators delete, merge, split, and rewrite skills. Without operator-level provenance, later audits cannot reconstruct which artifact contributed to a decision. Individually acceptable skills can be harmful when chained. ASG-SI [23] is the closest surveyed system to graph-level audit; SkillOrchestra gestures toward typed composition; neither closes the full provenance-plus-composition safety loop.

Six governance primitives. A defensible deployment needs (i) an admission-time scanner combining static checks, LLM/rubric analysis, sandboxing, and repository provenance; (ii) a package manifest for dependencies, permissions, model artifacts, network endpoints, allowed tools; (iii) a runtime containment layer for stdout, filesystem, network, bundled-model channels; (iv) an operator-granular provenance log for Add, Refine, Merge, Split, Prune; (v) a policy and domain release gate; (vi) a market governance layer for ownership, takedown, remediation, and abandoned-repository hijacking.

Table 13. Eight safety surfaces specific to dynamic-skill stores.

Surface	Quantitative anchor	Lifecycle control
1. Skill-file injection	ClawSafety 69.4% ASR	Parse, scan, verify before admission
2. Supply-chain poisoning	Supply-Chain 11.6–33.5% bypass; BadSkill 97.5–99.5% ASR	Sandboxing, signatures, dependency audit
3. Harmful-but-valid skills	HarmfulSkillBench 4.93% (4,858/98,440)	Scope, permissions, policy-aware retrieval
4. Credential leakage	Credential Leakage 76.3% NL+code; 73.5% debug logging	Static scanning + release gates
5. Scanner limits / false positives	SkillSieve 0.800 F1 / $0.006; Mal-or-Not 0.52% remaining	Repository context + provenance
6. Ownership / theft	SkillStealing few-query extraction	Access control, telemetry, provenance
7. Domain release	MedSkillAudit 57.3% below release; ICC 0.449	Domain-specific rubrics
8. Attribution & composition	ASG-SI, SkillOrchestra (partial)	Operator-level provenance

L Eight Open Problems with Evidence-Basis Tags

The first four problems improve algorithms inside existing mechanism families. The last four require community infrastructure for portability, governance, and trajectory-aware benchmarks. Method papers and infrastructure papers should therefore be judged together: dynamic skills will not mature if algorithms improve inside isolated libraries while portability, provenance, and benchmark realism lag behind.

M Operator Vocabulary Reference

Table 15. Ten-operator vocabulary for library transitions. This is a taxonomy for comparing systems, not a claim of algebraic closure or associativity.

Op.	Library transition	Typical evidence	Main risk
Add	Insert a new artifact.	Successful trace, demonstration, generated code.	Bloat and unverified candidates.
Refine	Patch an existing artifact in place.	Failure replay, unit test, user correction.	Overfit to one incident.
Merge	Combine overlapping artifacts.	Duplicate retrievals, shared subgoals.	Loss of specialized preconditions.
Split	Factor a broad skill into scoped pieces.	Conflicting contexts or long procedures.	Fragmentation, routing burden.
Prune	Remove or quarantine low-value entries.	Low use, failed verification, security scan.	Deleting rare but important skills.
Distill	Transfer artifacts into compact model behavior.	Repeated skill use, curated trajectories.	Loss of auditability.
Abstract	Convert traces into reusable rules / templates.	Repeated local lessons.	Overgeneralization.
Compose	Build a higher-order workflow from skills.	Dependency graph, planner output.	Hidden incompatibilities.
Rewrite	Change representation or interface.	Runtime migration, API change, package format.	Breaking callers.
Rerank	Change retrieval priority without changing content.	Usage logs, benchmark utility, recency.	Popularity bias, stale winners.

A single update may combine several operators. For example, a failed workflow can trigger Refine on one step, Prune on a stale dependency, and Rerank on the retrieval index. The library transition therefore decomposes as

$$\mathcal{L}_{t+1} = \mathrm{Admit}_{\nu_t}(\mathrm{Propose}_{\varphi_t}(\tau_t, r_t),\ \mathcal{L}_t),$$

followed by optional maintenance and distillation:

$$\mathcal{L}'_{t+1} = \mathrm{Maintain}(\mathcal{L}_{t+1}), \quad \theta_{t+1} = \mathrm{Distill}(\theta_t, \mathcal{L}'_{t+1}).$$

N Minimal Skill-Record Metadata Schema

A stored skill should carry enough metadata that a reader can tell what was stored, why it was admitted, when it was used, and how it can be repaired or removed.

Table 16. Minimal metadata for a lifecycle-managed skill artifact.

Field	Purpose
Skill ID and version	Identify the artifact and distinguish revisions.
Artifact type	Code, procedure, graph, package, policy, memory abstraction, or distilled target.
Interface	Inputs, outputs, runtime assumptions, dependencies, permissions.
Applicability context	Conditions under which retrieval or execution is intended.
Source evidence	Trace, demonstration, benchmark failure, user correction, prior skill.
Verifier record	Tests, replay, judge scores, static scans, human review, known gaps.
Admission decision	Accepted, rejected, quarantined, superseded, rolled back.
Lineage	Parents, merged sources, split children, distilled targets, rollback handle.
Usage statistics	Retrieval count, execution success, failures, stale periods, downstream tasks.
Safety metadata	Secrets scan, dependency audit, policy scope, owner, sharing boundary.

This schema is not a universal package standard. Its purpose is to make claims about library evolution reproducible: another reader should be able to run the same admission gate, reproduce the same retrieval, and roll back to the prior version.