They Are Not Static: A Survey of Dynamic Agent Skills

Anonymous *anonymous@example.com*
Anonymous Institution

Abstract

Large language model agents increasingly externalize procedural knowledge into reusable *skills*: invocable code, natural-language procedures, SKILL.md packages, graphs, or parametric adapters. This externalization turns adaptation into a new learning problem. The agent does not only update its prompt or weights; it updates a library of artifacts that changes what future policies can retrieve, compose, execute, and trust. This survey studies the rapidly growing 2023–2026 literature on *dynamic* or *self-evolving* skill systems and argues that such systems are best understood as *lifecycle-managed, verified, evolving artifact stores* for LLM agents. We extend the options-based skill formalism to a seven-tuple $\langle C, \pi, T, R, \varphi, \nu, \prec \rangle$ that makes edits, admission verification, and provenance explicit, and we lift it to library-level dynamics $\mathcal{L}_t \to \mathcal{L}_{t+1}$ driven by a ten-operator algebra (Add, Refine, Merge, Split, Prune, Distill, Abstract, Compose, Rewrite, Rerank). Using this formalism, we organize a 94-paper modern audit set of dynamic-skill and boundary/context papers around a skill lifecycle: evidence acquisition, proposal, verification/admission, organization, retrieval/composition, maintenance/repair, distillation/portability, and governance. The resulting taxonomy separates artifact families, update loci, assurance models, storage topologies, maintenance regimes, and governance maturity without reducing the field to a list of systems. We then synthesize the mechanisms that make lifecycle-managed stores improve: edit repertoires, admission gates, storage/retrieval structure, and fast–slow update clocks. The most consistent evidence is that admission and repair matter more than raw skill count, verifier quality is often load-bearing in skill-aware RL, flat retrieval degrades in the moderate-library-size regime, and benchmarks still under-report library trajectories. We close with a research agenda for compositional verifiers, maintenance schedules, registry-scale retrieval, cross-library portability, provenance, and lifecycle-aware evaluation.

1 Introduction

Large language model 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, coordinate multi-step research pipelines, interact with embodied environments, and assist domain-specific workflows such as recommendation and medical imaging. Across these settings, the bottleneck is not only whether the model can reason about a single task. It is whether the agent can reuse procedural knowledge across recurring tasks, tools, interfaces, and users instead of re-deriving the same strategy inside every context window.

Agent skills were proposed as a practical answer to this reuse problem. A skill is a modular, externally invocable procedural knowledge package: it may be a function, a natural-language procedure, a SKILL.md directory, a workflow graph, or a learned adapter, but its role is the same — to preserve a reusable way of acting so that a future agent can retrieve, compose, and execute it. Early and recent systems use skills to improve efficiency, reliability, and transfer across game environments, web agents, software-engineering agents, computer-use agents, multimodal agents, recommendation systems, and medical-imaging workflows (Wang et al., 2023; Zheng et al., 2025a; Xia et al., 2025a; Chen et al., 2026c; Jiang et al., 2026a; Tao et al., 2026; Fan et al., 2026). The broader skill ecosystem now includes SKILL.md conventions, function-calling schemas, MCP/plugin-style packaging, and registry-like platforms that support libraries with hundreds or thousands of reusable artifacts (Jiang et al., 2026b; Li et al., 2026c; Zheng et al., 2026).

This adoption changes the research question. In OpenClaw-style personal agents,[1] skills are no longer helpful prompt snippets; they are part of the execution substrate. SkillClaw uses cross-user OpenClaw interactions to evolve shared skills over deployment rounds (Ma et al., 2026), while ClawSafety shows that skill files can become a high-trust prompt-injection channel in privileged personal-agent scaffolds (Wei et al., 2026). The same abstraction that improves reuse creates questions of verification, maintenance, provenance, and governance.

Many skill libraries are still treated as *static*: authored once, versioned rarely, and weakly connected to the trajectories that reveal whether a skill remains useful, safe, or correct. Recent work (Alzubi et al., 2026; Zheng et al., 2025a; Wang et al., 2025; Ni et al., 2026; Shi et al., 2026; Li et al., 2026b; 2025; Xia et al., 2026b) rejects this assumption by treating the library itself as a learning object that grows, shrinks, repairs itself, changes storage structure, and sometimes distills its contents back into model weights. The methods differ—retrospective consolidation, execution-grounded honing, RL-based proposal, rollback-validated refactoring, registry-scale retrieval, and two-timescale distillation—but share the commitment that skill libraries should be managed as evolving systems.

The central thesis of this survey is that dynamic skill systems are *lifecycle-managed, verified, evolving artifact stores for LLM agents*. A skill artifact is created from evidence, proposed as an edit, verified, admitted, organized, retrieved or composed, maintained, sometimes distilled, and governed through provenance and rollback. Papers differ mainly in which stages they implement, which learning signal drives edits, and where they place the verifier.

1.1 Dynamic skill libraries as learning systems

We adopt the options-based formulation of Sutton et al. (1999) and its skill-level adaptation in SoK-Skills (Jiang et al., 2026b) as our starting point. A *skill* is a 4-tuple $\langle C, \pi, T, R \rangle$ over an applicability condition, an executable policy, a termination criterion, and a reusable interface. A *library* is a set of such skills equipped with a retrieval or composition mechanism. A *dynamic* library is a time-indexed sequence $\mathcal{L}_t$ in which each transition $\mathcal{L}_t \to \mathcal{L}_{t+1}$ is explained by a finite operator algebra applied to evidence drawn from the agent's trajectories. The precise formalism, including the 7-tuple extension that the survey's analysis requires, is developed in §3; the lifecycle architecture is developed in §5.

This makes dynamic skills a learning-systems problem rather than merely a software-packaging problem. Interaction produces evidence; evidence selects edits; edits change the library; the changed library alters the distribution of future actions through retrieval, composition, and execution. In this view, $\mathcal{L}_t$ is part of the agent's state, the transition $\mathcal{L}_t \to \mathcal{L}_{t+1}$ is a learning rule, verification is a selection mechanism, maintenance is a form of regularization, and distillation is a consolidation step that moves external procedural knowledge into slower parametric or semi-parametric stores. The same library can also serve as the interface between individual learning and collective learning when skills are shared across users, registries, or agents.

The time index matters because deployed libraries go stale as tasks, tools, backbones, and safety constraints drift. It also makes the empirical regularities in §9 expressible: admission and verifier quality, retrieval knees, weaker-backbone gains, focused curation, maintenance at scale, and write-time abstraction are all properties of a changing library.

1.2 Why a survey now

Three developments in 2025–2026 made dynamic skills surveyable. First, packaging and registry infrastructure matured enough that libraries must be curated, routed, and governed as collections (§10.3). Second, methods now span the learning-signal spectrum, from LLM-as-judge admission through audited verification and two-timescale RL. Third, deployment and benchmark papers report failure modes—retrieval degradation, skill injection, verification drift, skill inflation, and maintenance-off collapse—that earlier method papers treated mostly in ablation. SkillFlow-Bench (Zhang et al., 2026h), for example, evaluates dis-

[1] We use OpenClaw, SkillClaw, and ClawSafety as research-artifact or evaluation-setting names from the cited papers; the survey relies on the papers' methods and measurements, not on product or platform claims.

covery, patching, reuse, and maintenance over sequential tasks and shows that high skill-use rates need not imply high utility.

The field still lacks shared vocabulary, formalism, and benchmark protocol. Papers use "skill" for at least six structurally different artifacts (§3); operators are named inconsistently; benchmarks report terminal performance rather than library trajectories (§8). The survey's role is to provide a common language before ranking is meaningful.

1.3 Contributions

This paper makes five contributions. *First*, it recasts dynamic skills as an external, editable learning-systems object and formalizes it with a 7-tuple $\langle C, \pi, T, R, \varphi, \nu, \prec \rangle$ plus library-level dynamics $\mathcal{L}_{t+1} = \text{Apply}(u_t(\tau_t, r_t), \mathcal{L}_t)$ (§3). *Second*, it introduces a lifecycle architecture spanning evidence acquisition, proposal, verification/admission, organization, retrieval/composition, maintenance/repair, distillation/portability, and governance (§5). *Third*, it identifies a ten-operator algebra {Add, Refine, Merge, Split, Prune, Distill, Abstract, Compose, Rewrite, Rerank} and uses it to organize the corpus into a lifecycle taxonomy (§6). *Fourth*, it synthesizes the core mechanisms that make dynamic stores improve: edit repertoire, admission, storage/retrieval, and fast–slow consolidation (§7). *Fifth*, it audits lifecycle-aware evaluation, then distills seven empirical regularities, eight safety surfaces, and eight open problems tied to concrete next experiments (§§8, 9, 11, 12).

2 Corpus, Scope, and Review Protocol

We separate the object of study from the process used to assemble it. The survey covers papers that treat skill *libraries* or skill-like external artifacts as dynamic objects: artifacts may be added, revised, merged, pruned, routed, distilled, transferred, or governed as the agent accumulates evidence. It excludes classical option discovery, generic tool-use benchmarking, and fine-tuning pipelines unless they produce an externally invocable skill artifact or directly evaluate such artifacts' lifecycle.

2.1 Inclusion Criteria

The working audit set contains 94 modern papers after the April 24, 2026 update: 2 from 2023, 17 from 2025, and 75 from 2026. This set includes the primary dynamic-skill method, benchmark, infrastructure, and safety cluster, plus boundary/context papers that define terminology, adjacent lifelong-agent evaluation, or neighboring self-evolution settings. Older classical-options and hierarchical-RL anchors are cited for comparison but are not counted in this modern audit set. We included papers from 2023–2026 when they satisfied at least one of three criteria: (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/context works — SoK-Skills (Jiang et al., 2026b), Xu & Yan (2026)'s 2026 position paper, MAGELLAN's autotelic curriculum mechanism (Gaven et al., 2025), the ELL/StuLife benchmark (Cai et al., 2025), and the Experience Compression Spectrum framework (Zhang et al., 2026f) — are retained as scope-setting evidence rather than as members of the primary method cluster.

Figure 2 visualizes the corpus at the level needed for a survey claim: a sparse 2023 foundation, no primary 2024 item after filtering, a 2025 transition wave, and a dense 2026 expansion into executable libraries, infrastructure, benchmarks, and safety.

2.2 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 for

papers discovered during snowballing. Candidate papers 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. Because the collection was assembled in an arXiv-heavy and still-moving area, we report the final audit set and inclusion frontier rather than a PRISMA exclusion flow. The survey does not claim an exhaustive count of every tool-use, memory, or HRL paper adjacent to the topic.

2.3 Temporal Cutoff and Update Policy

All claims in the main text should be read against an explicit temporal cutoff: April 24, 2026. Papers appearing after that date are outside the surveyed corpus. The cutoff is an audit boundary, not a claim that the field has stabilized. We distinguish *corpus updates*, which add papers satisfying the criteria above, from *claim updates*, which revise the taxonomy, regularities, or open problems only when new evidence changes a lifecycle stage or exposes a new failure mode.

2.4 Note Schema and Conflict Resolution

Each included paper was read into a common note schema covering problem framing, mechanism, control action, experimental setup, headline results, ablations, limitations, closest peers, and survey takeaway. When duplicate notes disagreed, we resolved the conflict against the PDF. The schema is important because several names are close but technically distinct: SkillFlow-2025 is a multi-stage retrieval system for community skill repositories, while SkillFlow-Bench is a lifelong skill-discovery benchmark; SAGE stores executable Python skills, while PSN's refactors are rollback-validated graph edits.

2.5 Evidence Roles

The corpus mixes method papers, infrastructure papers, benchmarks, and safety studies. We use method papers for claims about operators, triggers, verifiers, and mechanisms; benchmark papers for evaluation protocol and lifecycle failure modes; infrastructure papers for storage, portability, registries, and operator cost; and safety papers for attack surfaces and partial defenses. We grade evidence qualitatively: **A** means multiple controlled ablations or one clean ablation plus independent corroboration; **B** means one controlled study or a strong benchmark/deployment measurement; **C** means convergent benchmark behavior without a clean causal ablation; and **D** means architectural corroboration only. We avoid cross-paper leaderboards unless the harness is shared, because backbone, tool surface, context budget, and evaluator often change together.

2.6 Relation to Adjacent Surveys

Three peer surveys cover adjacent territory and we do not duplicate their full scope. Xu & Yan (2026) surveys agent-skill architecture, acquisition, and security from a broad position-paper vantage; Fang et al. (2025) surveys self-evolving agents beyond skill libraries; and Zheng et al. (2025c) surveys lifelong learning for LLM agents with a memory-centric emphasis. Our contribution relative to these is the library-as-object formalism, the lifecycle architecture, and the operator-level taxonomy: instead of asking whether agents can learn over time in general, we ask how an externally invocable artifact store changes, verifies, maintains, and governs itself.

The scope also separates this survey from classical options and hierarchical reinforcement learning. Options, skill chaining, option-critic methods, and deep skill-discovery methods study temporally extended policies inside an environment (Sutton et al., 1999; Konidaris & Barto, 2009; Bacon et al., 2017; Eysenbach et al., 2019; Nachum et al., 2018). Dynamic agent skills add a different object: an externally inspectable artifact that can be edited after deployment, packaged with metadata, admitted or rejected by language- or execution-level verifiers, shared across agents, and governed through provenance. We use the options formalism as a

starting point because it clarifies applicability, policy, and termination; the survey's added components are the artifact-store machinery that classical HRL usually leaves implicit.

3 Background and Skill Formalism

The phrase "agent skill" now denotes at least six different artifacts: executable programs (Voyager, SAGE) (Wang et al., 2023; 2025), verified web procedures (SkillWeaver) (Zheng et al., 2025a), SKILL.md packages (MetaClaw, ABSTRAL, Trace2Skill, Memento, SkillFlow-Bench) (Xia et al., 2026b; Song et al., 2026; Ni et al., 2026; Zhou et al., 2026; Zhang et al., 2026h), multimodal skill documents plus experience banks (XSkill) (Jiang et al., 2026a), parametric adapters (SkillsCrafter) (Wang et al., 2026d), registry-retrieved SKILL.md candidates (SkillFlow-2025) (Li et al., 2025), and capability labels (Co-Evolving-Agents) (Jung et al., 2025). A single undifferentiated definition would either collapse these artifacts or become too abstract to support comparison.

We therefore separate *terminology* from *formalism*: the terminology distinguishes artifact types, while the formalism extends options-style skills (Sutton et al., 1999; Jiang et al., 2026b) with the minimal machinery needed to describe edits, verification, lineage, and library-level change.

3.1 The six senses of "skill"

Table 1 partitions the literature along five dynamic properties: whether the artifact is *executable*, *editable in place*, *portable across models*, *inspectable by humans*, and attached to a *verification handle*. These properties largely determine which triggers, operators, and signals a method can support.

The executable-code sense, occupied by Voyager, LATM, LIVE-SWE, AgentFactory, and SAGE (Wang et al., 2023; Cai et al., 2023; Xia et al., 2025a; Zhang et al., 2026g; Wang et al., 2025), treats a skill as a named program that can be invoked, inspected, edited line-by-line, and verified by tests or environment feedback. The NL-heuristic sense, occupied by ERL, RetroAgent, and EvolveR (Allard et al., 2026; Zhang et al., 2026e; Wu et al., 2025), treats a skill as a short lesson injected at retrieval time; it is editable and portable but only indirectly verifiable through downstream task success. The SKILL.md sense, occupied by MetaClaw, ABSTRAL, Trace2Skill, Memento, K2-Agent, AutoRefine, and SkillFlow-Bench (Xia et al., 2026b; Song et al., 2026; Ni et al., 2026; Zhou et al., 2026; Wu et al., 2026b; Qiu et al., 2026; Zhang et al., 2026h), unifies code and prose behind a progressive-disclosure package: L1 metadata for routing, L2 instructions for the agent, and L3 code, schemas, or sub-skills for execution and verification.

The parametric sense, represented by SkillsCrafter, SELAUR, SKILL0, and LSE (Wang et al., 2026d; Zhang et al., 2026b; Lu et al., 2026; Chen et al., 2026d), treats a skill as a weight delta, trained prompt module, or internalized procedure; such artifacts are probed behaviorally, ported only across compatible base models, and edited through training. The memory/trajectory sense, present in SimpleMem, MUSE, CASCADE, XSkill, and the case layer of Memento (Liu et al., 2026b; Yang et al., 2025a; Huang et al., 2025; Jiang et al., 2026a; Zhou et al., 2026), blurs skills with retrievable episodes; the boundary is whether the trace has an invocable interface. The registry-retrieved and capability-label senses, present in SkillFlow-2025, Co-Evolving-Agents, and some of GEA (Li et al., 2025; Jung et al., 2025; Weng et al., 2026), are the most brittle under dynamic evolution because retrieval success or claimed capability has no robust edit body unless paired with executable skill contents and admission checks.

In the rest of the paper, "skill" means executable code, NL heuristic, SKILL.md package, or parametric skill by default; memory/trajectory and capability-label artifacts are treated as boundary cases.

This definition also separates *skills* from ordinary *tools*. A tool is usually an externally supplied callable resource: the agent decides when to invoke it, but the tool's body, interface, and release process are not learned by the agent. A dynamic skill is a persistent artifact whose lifecycle is itself part of learning. The same Python function or MCP endpoint can therefore be a tool in a generic tool-use benchmark and a skill in a dynamic-skill system if the agent proposes it, revises it from trajectories, verifies it for admission,

stores lineage, and later maintains or transfers it. The distinction is operational rather than syntactic: what matters is whether the artifact participates in $\mathcal{L}_t \to \mathcal{L}_{t+1}$.

Method	Sense of "skill"	Artifact form	Exec.	Edit	Portable	Inspect.	Verif. handle
Voyager (Wang et al., 2023)	Executable code	.py library	✓	✓	∼[a]	✓	unit test
LATM (Cai et al., 2023)	Executable code	Python tool	✓	✓	∼[a]	✓	unit test
AgentFactory (Zhang et al., 2026g)	Executable code	Python + MCP	✓	✓	∼[a]	✓	meta-agent inspect.
LIVE-SWE (Xia et al., 2025a)	Executable code	code snippet	✓	✓	∼[a]	✓	rollout
SAGE (Wang et al., 2025)	Executable code	Python function	✓	✓	∼[a]	✓	env. reward
ERL (Allard et al., 2026)	NL heuristic	critique	–	✓	✓	✓	indirect
RetroAgent (Zhang et al., 2026e)	NL heuristic	dual lesson	–	✓	✓	✓	indirect
EvolveR (Wu et al., 2025)	NL heuristic	strategic principle	–	✓	✓	✓	indirect
MetaClaw (Xia et al., 2026b)	SKILL.md	L1/L2/L3 markdown	✓[b]	✓	✓	✓	L3 code gate
Memento (Zhou et al., 2026)	SKILL.md + case	markdown + trace	✓[b]	✓	✓	✓	L3 code gate
Trace2Skill (Ni et al., 2026)	SKILL.md	L2 markdown	✓[b]	✓	✓	✓	rubric judge
AutoRefine (Qiu et al., 2026)	SKILL.md	dual-form markdown	✓[b]	✓	✓	✓	rubric judge
SkillFlow-Bench (Zhang et al., 2026h)	SKILL.md patch	files + JSON patch	✓[b]	✓	✓	✓	benchmark verifier
ABSTRAL (Song et al., 2026)	SKILL.md / design doc	structured markdown	–	✓	✓	✓	trace evidence
SkillsCrafter (Wang et al., 2026d)	Parametric	LoRA subspace	✓[c]	–	–	–	behavioral probe
SKILL0 (Lu et al., 2026)	Parametric	adapter	✓[c]	–	–	–	behavioral probe
SELAUR (Zhang et al., 2026b)	Parametric	LoRA	✓[c]	–	–	–	behavioral probe
LSE (Chen et al., 2026d)	Parametric (mix)	prompt + weight	✓[c]	∼	–	∼	behavioral probe
SimpleMem (Liu et al., 2026b)	Memory / trajectory	trace case	–	∼	✓	✓	indirect
MUSE (Yang et al., 2025a)	Memory / trajectory	episodic store	–	∼	✓	✓	indirect
CASCADE (Huang et al., 2025)	Memory / trajectory	curated case set	–	∼	✓	✓	indirect
XSkill (Jiang et al., 2026a)	Skill + experience	MD skill + JSON exp.	∼	✓	✓	✓	indirect
SkillFlow-2025 (Li et al., 2025)	Registry-retrieved skill	SKILL.md corpus + ranked candidates	∼	–	✓	✓	retrieval eval
Co-Evolving (Jung et al., 2025)	Capability label	declarative role	–	∼	✓	✓	none
GEA (Weng et al., 2026)	Capability label	declarative role	–	∼	✓	✓	none

Table 1: **Representative dynamic-skill methods keyed by paper, with the sense of "skill" each adopts and the five properties that determine its dynamic behaviour.** The six senses partition the design space (horizontal rules) and project directly onto the taxonomy axes of Section 6; the first four senses are the "dynamic skills" studied in this survey while the last two are boundary cases with restricted edit / verification dynamics. Column definitions: *Exec.* = is the artifact machine-executable; *Edit* = can be revised without retraining; *Portable* = usable across different LLM backbones; *Inspect.* = auditable in textual form; *Verif. handle* = form of admission check available. Symbols: ✓ = fully satisfies; ∼ = partial / indirect; – = does not satisfy. [a]portable across backbones that can call the same runtime; [b]executable only when L3 attaches code or MCP resources; [c]executable only through the compatible base model.

3.2 From static options to a 7-tuple skill

Starting point. The canonical reference for skills as temporally extended actions is the options framework of Sutton et al. (1999): an option is a triple $\langle I, \pi, \beta \rangle$ of an initiation set $I \subseteq \mathcal{S}$, a policy π, and a termination condition β. Within the LLM-agent literature, the systematization-of-knowledge paper by Jiang et al. (2026b) reinterprets this triple for language agents as a four-tuple

$$\mathcal{S} \;=\; \langle C,\ \pi,\ T,\ R \rangle, \tag{1}$$

where C is an applicability predicate ("when is this skill relevant"), π is the executable policy (code, prompt template, adapter, or lesson text), T is a termination condition (success, exception, or budget exhaustion), and R is a *reusable interface* (a typed set of inputs, outputs, and invocation points) that lets another program or skill compose the artifact.

Five concrete gaps. The SoK four-tuple is adequate for *static* libraries, but dynamic libraries require five missing objects. First, a *time index* distinguishes $\mathcal{S}_t$ from later refinements, replacements, and merges. Second, an *edit operator* records whether a method rewrites NL lessons (ERL, EvolveR, RetroAgent), function bodies (SAGE), SKILL.md prose (AutoRefine, Memento), symbolic refactors (PSN) (Shi et al., 2026), or mutation heuristics (CODE-SHARP) (Bornemann et al., 2026). Third, an *admission gate* captures filters such as EvoSkill's Pareto front (Alzubi et al., 2026), SkillCraft's MCP verifier (Chen et al., 2026b), and ASG-SI's audited graph (Huang & Huang, 2025). Fourth, *lineage* supports rollback, supersession, and maturity gating in systems such as AgentDevel, PSN, Memento, and Trace2Skill (Zhang, 2026; Shi

et al., 2026; Zhou et al., 2026; Ni et al., 2026). Fifth, a *library-level object* is needed because the main transition is $\mathcal{L}_t \to \mathcal{L}_{t+1}$, not merely one tuple to another.

The extended tuple. We therefore extend Equation 1 to the seven-tuple

$$\mathcal{S}_t \;=\; \langle\, C_t,\ \pi_t,\ T_t,\ R_t,\ \varphi_t,\ \nu_t,\ \prec_t \,\rangle, \tag{2}$$

with the new components interpreted as follows. The *edit operator* φ_t is the candidate-generation rule for a skill revision under an edit instruction u_t (chosen from the operator algebra introduced below in §3.3). In deterministic systems, $\varphi_t : \mathcal{S}_t \times u_t \to \mathcal{S}_{t+1}$ maps directly to a successor skill. In stochastic proposal systems such as mutation-based search, φ_t is better read as a proposal kernel $q_t(\mathcal{S}' \mid \mathcal{S}_t, u_t)$, from which a realized candidate $\tilde{\mathcal{S}}_{t+1}$ is sampled before admission. The edit component therefore need not guarantee improvement; the admission predicate below decides whether the realized candidate changes library state. The *verification predicate* $\nu_t : \mathcal{S}_t \to \{0, 1\}$ decides admissibility—whether a proposed or edited skill passes the quality gate and is allowed into $\mathcal{L}_{t+1}$—and may be a unit test (LATM), a grounded rollout (SkillWeaver, EvoSkill), a meta-agent inspection (AgentFactory), an ensemble judge (Trace2Skill, AgentSkillOS), a static analyzer (SkillCraft), a symbolic audit (ASG-SI), or a Bayesian prior over future utility (CODE-SHARP). The *lineage relation* $\prec_t$ is a partial order that records which version supersedes which; methods that support rollback, A/B maturity, or blast-radius limits require an explicit $\prec$.

Static libraries are recovered by setting $\varphi_t \equiv \mathrm{id}$, $\nu_t \equiv 1$, and $\prec_t \equiv \varnothing$, so Equation 2 strictly contains Equation 1. The artifact type determines feasible $(\varphi, \nu, \prec)$ choices: executable skills support machine-checkable gates, NL heuristics require indirect verification, parametric skills move edits onto a training timescale, and capability labels usually lack a well-defined edit operator. Cross-skill operations such as composition, merging, and splitting are library-level transitions, so we formalize them next.

3.3 Library dynamics: $\mathcal{L}_t \to \mathcal{L}_{t+1}$

A dynamic skill system is a library plus rules for how that library changes. Write

$$\mathcal{L}_t \;=\; \{\mathcal{S}_t^{(1)}, \mathcal{S}_t^{(2)}, \ldots, \mathcal{S}_t^{(N_t)}\} \cup \mathcal{M}_t, \tag{3}$$

where $\mathcal{M}_t$ holds auxiliary metadata: invocation statistics, call graphs, maturity labels, verifier caches, and any cross-skill edges (prerequisite, composition, shared resources, author). The library transition is then

$$\mathcal{L}_{t+1} \;=\; \mathcal{T}\big(\mathcal{L}_t,\ \tau_t,\ r_t\big) \;=\; \mathrm{Apply}\big(\vec{u}_t(\tau_t, r_t),\ \mathcal{L}_t\big), \tag{4}$$

where τ_t is an *evolution trigger* (a timer, a task boundary, a failure event, a user edit), r_t is the *learning signal* the system uses to choose an edit (task reward, natural-language critique, self-judgment, cross-user aggregate, teacher signal), and $\vec{u}_t$ is a vector of operator-instruction pairs drawn from the ten-element algebra

$$\vec{u}_t \subseteq \{\textsc{Add}, \textsc{Refine}, \textsc{Merge}, \textsc{Split}, \textsc{Prune}, \textsc{Distill}, \textsc{Abstract}, \textsc{Compose}, \textsc{Rewrite}, \textsc{Rerank}\} \times \mathrm{Instr}.$$

The ten operators have fixed meanings throughout the paper: Add inserts a skill; Refine edits content without changing the interface; Merge combines skills; Split factors one skill into components; Prune removes or quarantines; Distill compresses trajectories into a skill; Abstract lifts a concrete procedure to a template; Compose chains skills into a composite; Rewrite changes the body and possibly the interface; and Rerank changes retrieval priors without changing content. Representative instances include Voyager and SAGE for Add, Memento and PSN for Refine, Trace2Skill and AutoRefine for Merge, SkillX for Split, Wild-Skills and ClawSafety (Wei et al., 2026) for Prune, CASCADE and MUSE for Distill, CUA-Skill and CoEvoSkills for Abstract, SkillCraft and SkillOrchestra for Compose, EvolveR for Rewrite, and SkillRouter for Rerank. Almost no method implements all ten; the supported subset is one of the clearest taxonomic fingerprints.

Verification as library-level gate. Although ν is indexed per skill in Equation 2, verification acts at *admission* time: an edit yields a candidate $\mathcal{S}^*$, and $\mathcal{S}^*$ enters $\mathcal{L}_{t+1}$ only if $\nu(\mathcal{S}^*) = 1$. This edit-versus-admission distinction underlies the verification architecture of Section 7 and the R1 regularity in Section 9.

Two timescales. The trigger τ_t can be a per-step failure (SELAUR), per-task retrospective pass (SAGE, ERL), periodic maintenance cycle (AutoRefine), or release decision (AgentDevel). Parametric systems such as MetaClaw, SKILL0, and LSE add a slow loop in which many fast-loop library updates are distilled into weights or adapters.

Scope. Throughout the rest of the paper, "dynamic skill" refers to a skill $\mathcal{S}_t$ whose $(\varphi, \nu, \prec)$ components are nontrivial or whose embedding library $\mathcal{L}_t$ evolves under a non-trivial $\vec{u}_t$. A *static* library is the special case $\vec{u}_t \equiv \varnothing$. The survey addresses the dynamic case; static-library methods are included only as baselines or as infrastructure substrates on which dynamic methods are built.

3.4 Worked instantiation: AutoRefine as a library transition

The notation is intended to describe concrete system behavior, not just to name components. Consider an AutoRefine-style maintenance cycle (Qiu et al., 2026). 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. After a trajectory exposes a failure or redundancy, the trigger is $\tau_t = \textsc{TaskEnd}$ or $\textsc{Periodic}$, and the signal r_t is a critique, execution trace, or downstream utility measurement. The update rule chooses an operator vector such as

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

Each realized edit yields a candidate $\mathcal{S}^*$. The admission gate evaluates $\nu_t(\mathcal{S}^*)$ 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. In this example, the transition in Equation 4 is not a single append operation: it is a gated composition of Refine, Merge, and Prune over a versioned artifact store.

4 Why Static Skill Libraries Fail

The formalism of §3 treats the skill library as a time-indexed object $\mathcal{L}_t$ because the static alternative fails in recurring, connected ways. Static libraries are useful when the task distribution, tool surface, verifier, and authoring assumptions remain stable. The surveyed literature shows that long-lived agent deployments rarely satisfy those conditions. The failure is not one defect but a lifecycle collapse: authoring, verification, retrieval, provenance, and adaptation are all forced into a one-time design decision.

The first pressure is economic. Static skills require up-front human authoring, yet deployment value is only observed after the skill has been used in context. Deployment-facing papers such as SkillClaw, AutoSkill, and AgentSkillOS describe useful SKILL.md authoring as labor-intensive, and SoK-Skills observes that library quality is bounded by the weakest authors. The issue is not simply that documentation is costly; it is that a static library pays the cost per skill before knowing which skills will matter. Dynamic systems shift part of that cost to write-time Abstract and Distill, so trajectory evidence decides what should become reusable.

The second pressure is that correctness is not stationary. A static library freezes the author's implicit verifier at authoring time, but tasks, tools, base models, runtimes, and safety constraints drift. PSN's refactor detectors depend on recent transition history; AgentSkillOS's capability-tree audits assume a current tool surface; CODE-SHARP's execution verifier assumes a compatible runtime. Once those assumptions move, a skill can remain syntactically valid while becoming operationally wrong. Dynamic systems make ν re-runnable and allow the admission gate to remove, quarantine, or demote skills whose verifier has drifted.

The third pressure appears as the library grows: selection becomes harder than storage. Flat retrieval exhibits a knee in the moderate-library-size regime, roughly one hundred skills. Single-Agent-Skills gives the cleanest controlled size curve, while Wild-Skills, SkillRouter, and AgentSkillOS show related degradation under realistic distractors, 80K-scale routing, and large flat stores (Li, 2026; Liu et al., 2026d; Zheng et al., 2026; Li et al., 2026b). The exact threshold varies, but the mechanism is stable: adding skills eventually adds distractors faster than utility. A static system can cap the library or redesign retrieval, but it has no native way to ask whether old skills should be merged, pruned, or reranked. Dynamic systems add those maintenance operators: Prune, Merge, and Rerank.

The fourth pressure is provenance. Ordinary version control records who edited a file and when, but not which trajectory, verifier, or deployment signal justified admission. That gap matters when a skill regresses (AutoRefine, AgentSkillOS), when admission must be re-run against a new verifier, or when a cross-user skill must be attributed, redacted, or rolled back (SkillClaw, AutoSkill). PSN's rollback gate makes the point concrete: tentative refactors are reverted if success on three recent tasks drops by more than 20%. The lineage relation $\prec$ is the minimal structural addition that makes rollback, re-admission, and provenance tractable.

Finally, deployment itself is non-stationary. A static library is a snapshot of the authoring distribution, but real task mixes shift. The surveyed evidence adds two important shapes to this familiar claim: weaker backbones gain disproportionately from dynamic skills, and focused libraries become stale when the task mix changes. The response is not merely to refresh the library occasionally; it is to specify an evolution trigger τ and a fast-loop clock that decide when evidence is allowed to alter the library.

These failures explain why the added components of the 7-tuple are not cosmetic. Authoring cost points to φ; verifier drift points to re-runnable ν; retrieval pollution points to maintenance operators; attribution loss points to $\prec$; and task non-stationarity points to explicit triggers. Table 2 reads the same mapping from the architecture side, and Table 11 maps methods to the lifecycle gaps they address.

5 Dynamic Skill Systems as Lifecycle-Managed Stores

Section 4 gives the negative case: static skill libraries fail because they make authoring, verification, retrieval, provenance, and adaptation one-time decisions. The positive object is therefore not just a larger skill library. It is a controlled state machine over an evolving store of artifacts. 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 time, maintains it as the library ages, and records enough provenance to support rollback, transfer, and governance.

This section defines that reference architecture. The next section uses it as a taxonomy for the surveyed papers; Section 7 then analyzes the implementation choices that make particular lifecycle stages possible. Keeping these roles separate is important: the lifecycle is the *architecture*; the taxonomy is the *classification of systems*; the mechanism design space is the *choice of operators, verifiers, and clocks*.

5.1 Reference Architecture

Figure 1 gives the high-level architecture; Table 2 records the corresponding design questions, operators, evidence, and failure modes. We decompose a dynamic skill system into eight recurring stages. *Evidence acquisition* decides what observation can justify a library change: a trajectory, reward, failure trace, user edit, cross-user signal, or external resource. *Proposal* converts that evidence into a candidate artifact or edit. *Verification and admission* decides whether the candidate is allowed to enter the library, and whether it enters as mature, tentative, quarantined, or rejected. *Organization and storage* assigns the admitted artifact to a flat index, hierarchy, DAG, invocation graph, ontology, dual memory/skill store, or parametric subspace. *Retrieval and composition* decides which artifacts influence a future action. *Maintenance and repair* keeps the library compact and correct by pruning, merging, splitting, reranking, refining, or rewriting artifacts. *Distillation and portability* moves procedural knowledge between external artifacts, model weights, agents, users, or task domains. *Governance and provenance* records lineage, authorship, safety checks, release state, and rollback handles.

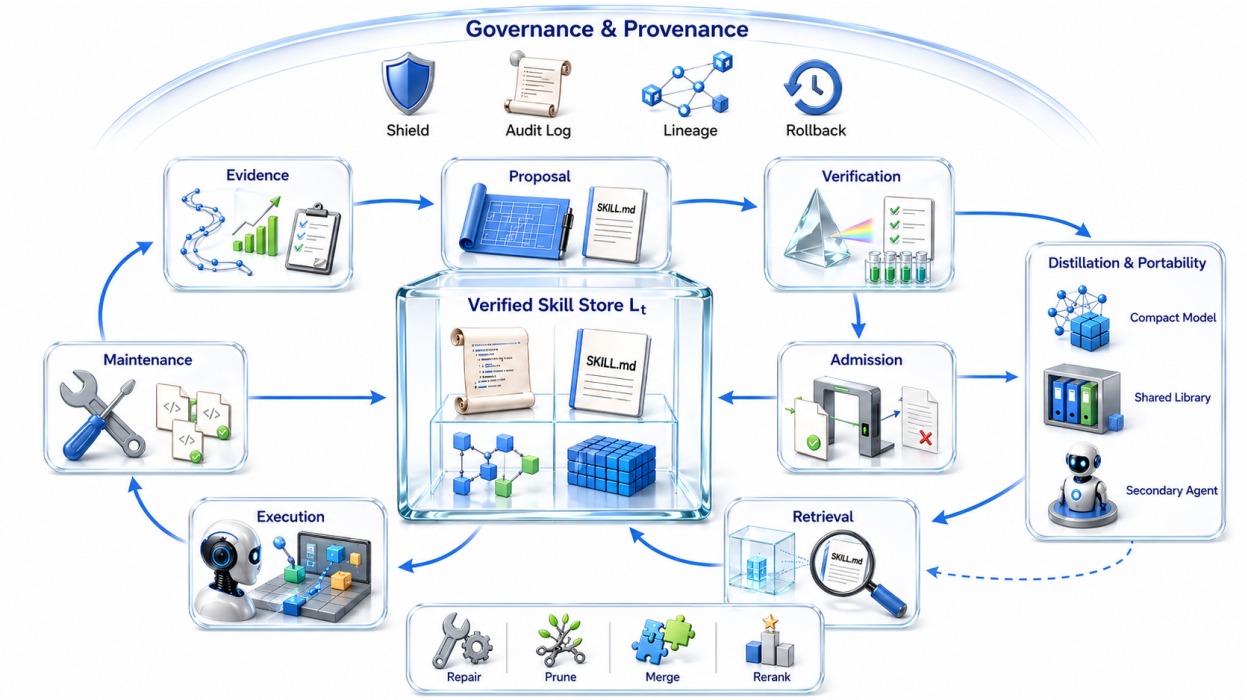

Figure 1: **Dynamic skill systems as lifecycle-managed artifact stores.** Interaction evidence drives proposal and verification; admitted artifacts enter an evolving skill store, where retrieval and execution create further evidence. Maintenance repairs, prunes, merges, or reranks the store over time. Governance and provenance wrap the lifecycle, while distillation and portability form a slower side loop. The illustration compresses the textual eight-stage lifecycle: organization/storage is represented by the central store, execution is the action point that produces new evidence, and governance/distillation are wrapper or side-loop functions rather than extra linear stages.

The stage order should not be read as a waterfall. Many systems loop between proposal and verification, perform retrieval before maintenance, or run distillation only periodically. The point is that each stage asks a different design question. A system that performs strong retrieval has not necessarily solved admission; a system that frequently uses a skill has not necessarily shown utility; a system that can add skills has not necessarily learned how to remove or repair them.

5.2 The Admission Boundary

The most important boundary in the lifecycle is between *candidate* artifacts and *admitted* library state. Dynamic systems can generate many plausible skills cheaply, but the library only improves when the admission gate filters them with an appropriate verifier. Execution-grounded systems such as SkillWeaver, EvoSkill, and SkillFoundry place the gate close to write time; maintenance-heavy systems such as AutoRefine and AgentSkillOS re-run the gate as the library ages; rollback systems such as PSN treat admission as tentative until recent-task utility remains stable. This boundary explains why verification is not a side module. It is the selection mechanism that turns skill generation into library learning.

Admission also determines what provenance must be stored. If a skill was admitted because of a trajectory, validator, cross-user signal, or release audit, the system must retain that justification so that future maintenance can demote, revise, or remove the artifact. Without this lineage, dynamic skills become append-only memories with a more polished file format.

Lifecycle stage	Design question	Main operators	Representative evidence	Characteristic failure
Evidence acquisition	What observation justifies a library change?	–	trajectories and rubrics in Trace2Skill, SkillFlow-Bench; visual rollouts in XSkill; rewards in SAGE, SkillRL, MACRO; failure tickets in SkillForge; external resources in SkillFoundry	noisy evidence, non-stationary utility
Proposal / externalization	How is evidence converted into an artifact?	Add, Abstract	retrospective lessons in ERL, RetroAgent; executable proposals in SkillWeaver, EvoSkill; file patches in SkillFlow-Bench; repository mining in SkillRepoMining; corpus compilation in Corpus2Skill	task-specific or overgeneral skills
Verification and admission	What gate decides whether the candidate enters?	Refine, Rewrite	execution checks in SkillWeaver; Pareto admission in EvoSkill, SkillMOO; surrogate tests in CoEvoSkills; multi-test validation in SkillFoundry; rollback validation in PSN; learned gates in CODE-SHARP	plausible but wrong skills; verifier hacking
Organization and storage	Where does the admitted artifact live?	Split, Merge, Rerank	capability trees in AgentSkillOS; prerequisite DAGs in CODE-SHARP; invocation graphs in PSN; ontology graphs in SkillNet; dependency retrieval in Graph of Skills; scoped packages in Skilldex	flat-retrieval collapse, stale structure
Retrieval and composition	Which skills affect the next action?	Rerank, Compose	SkillRouter's 80K routing; SkillFlow-2025's multi-stage community-skill retrieval; Wild-Skills' realistic-retrieval protocol; Graph of Skills' dependency-aware retrieval; GraSP' typed DAG composition; XSkill's visual task adaptation; SkillOrchestra's typed routing	usage without utility, distractor load
Maintenance and repair	How is the library kept compact and correct?	Prune, Merge, Refine, Rewrite	AutoRefine's pruning/merging ablation; XSkill's dual-store consolidation; AgentSkillOS's audits; PSN's maturity gating; SkillForge's failure diagnosis; skill inflation in SkillFlow-Bench	unbounded growth, negative transfer
Distillation and portability	What moves between external artifacts, weights, and agents?	Distill, Abstract	MetaClaw, K2-Agent, SKILL0, SkillsCrafter; cross-agent reuse in CoEvoSkills and XSkill	parametric collapse, compatibility failure
Governance and provenance	Can edits be audited, attributed, and rolled back?	logged Add–Prune sequence	audited graph in ASG-SI; release workflow in AgentDevel; threat taxonomy in Secure-Skills; scanners in SkillSieve; registry studies in Credential Leakage, Malicious-or-Not; release audit in MedSkillAudit	skill injection, leakage, attribution loss

Table 2: **Lifecycle architecture for dynamic skill systems.** The table is not a method ranking; it identifies the recurring stages at which a system must make design commitments. Later tables specialize this architecture into system families, verification architectures, empirical regularities, and open problems.

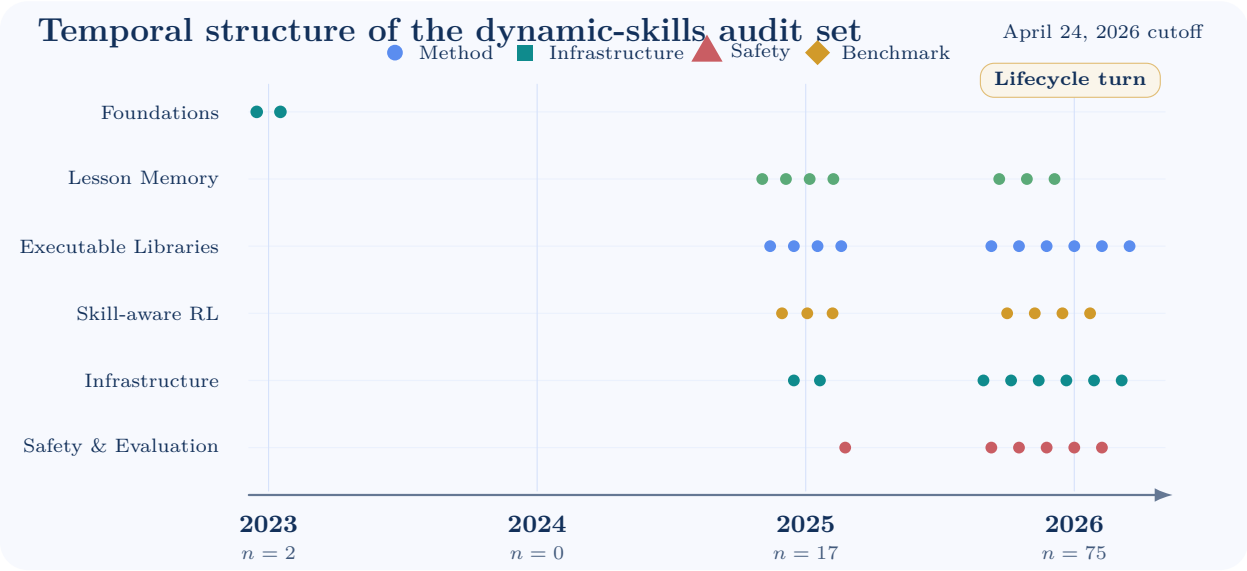

Figure 2: **Temporal structure of the dynamic-skills audit set.** Counts summarize the 94 modern papers covered by the survey and exclude only older classical-options and HRL background anchors; boundary/context papers are included for scope but are not treated as primary causal evidence in the regularities. Lane positions are schematic family locations rather than a stacked-count plot. The figure makes the survey's temporal claim explicit: the field becomes surveyable when 2025–2026 methods expand from early skill libraries into lifecycle mechanisms, registry infrastructure, benchmarks, and safety studies.

5.3 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.

This distinction sets up the taxonomy in Section 6. The lifecycle table above defines the design commitments a complete dynamic skill system must make; the taxonomy asks which subsets of those commitments each paper actually implements.

6 A Lifecycle Taxonomy of Dynamic Skill Systems

The taxonomy asks which lifecycle configuration a system instantiates. We use two levels: lifecycle families for the conceptual map, and seven coding fields in the master coding sheet (Table 11, Appendix A) for auditability.

6.1 Primary Families

Table 3 separates systems that are often conflated. Retrospective lesson systems and PPVH systems both update after a task, but only PPVH has execution-grounded admission. Skill-aware RL and two-timescale systems both use reward, but only the latter separates fast external edits from slow parametric internalization. Cross-user transfer and registry-scale infrastructure both handle many skills, but one concerns distributed authorship and the other routing/storage. Benchmarks such as SkillFlow-Bench and Wild-Skills are included because they expose lifecycle behavior method papers often hide.

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

Table 3: **Primary lifecycle taxonomy of dynamic skill systems.** Rows are families, not rankings. The table compresses each family into its dominant lifecycle stages, typical operator footprint, assurance bottleneck, and residual failure mode. This makes the comparison sharper than a per-paper list: families differ mainly in which parts of $\mathcal{L}_t \to \mathcal{L}_{t+1}$ they make cheap, verified, or governable.

6.2 Seven Coding Fields and Three Couplings

We code each representative system along seven fields: *artifact type* (code, NL lesson, SKILL.md package, memory trace, graph/workflow, or weight delta), *update locus*, *evolution trigger*, *operator repertoire*, *learning signal*, *storage topology*, and *model portability*. These are not claimed to be orthogonal basis dimensions. They are audit fields whose couplings are often the point.

Three couplings matter most. *Artifact–verifier coupling*: executable code supports tests and rollouts, while NL lessons and capability labels rely on weaker downstream or judge-based checks. *Storage–maintenance coupling*: flat stores make Add cheap but make Merge and Prune costly; graphs and hierarchies expose structure but require more careful rollback. *Trigger–signal coupling*: task-end retrospection naturally supplies critique, RL loops supply reward, user edits supply ownership constraints, and deployment registries supply aggregate utility. This dependence is why Table 3 is the conceptual taxonomy and Table 11 is the coding sheet. Figure 3 gives the corresponding visual summary: families differ less in whether they "use skills" than in which lifecycle stages they actually cover.

6.3 Master Coding Table

Table 11 in Appendix A instantiates the seven coding fields for representative systems. The "headline" column is factual rather than evaluative so that the table remains a map, not a comment collection. We place the full coding sheet in the appendix because it is an audit artifact; the main text uses Table 3 and Figure 3 for synthesis.

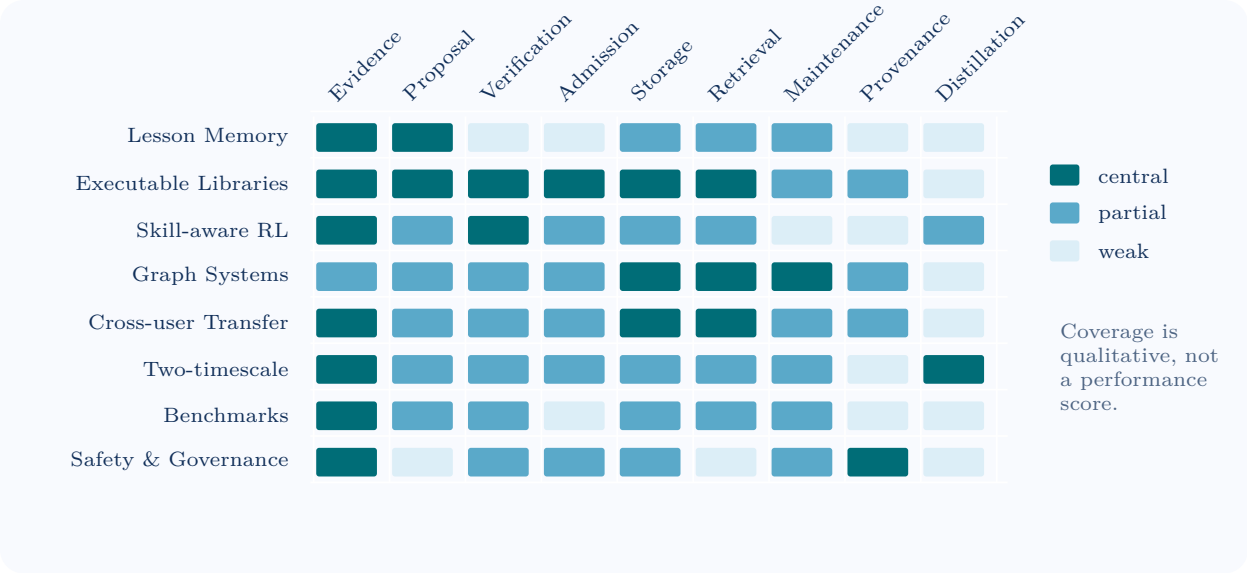

Figure 3: **Lifecycle coverage across system families.** Cell intensity summarizes how centrally a family implements a lifecycle stage in the coding sheet. The heatmap is a synthesis device, not a ranking: executable libraries concentrate around proposal, verification, admission, storage, and retrieval; graph systems concentrate around storage and maintenance; two-timescale systems concentrate around distillation; and safety/governance systems concentrate around provenance.

6.4 What the Taxonomy Reveals

Three diagonals matter most. Artifact and storage co-vary; trigger and signal co-vary; and lifecycle maturity is visible from the operator set. The newest infrastructure and safety papers add a fourth diagonal: once skills are packaged for registries, scanners, package managers, and repository-context checks become part of the same taxonomy as routing and maintenance. These diagonals motivate the mechanism synthesis of §7: operator algebra, verification architecture, and fast–slow adaptation.

7 Mechanisms: How Dynamic Skill Stores Improve

Sections 5–6 define the object of study and classify the corpus. This section asks a narrower question: what mechanism turns experience into a better skill store? Across the literature, four choices are load-bearing. A system must decide which edits it can make, how candidate edits are admitted, how the resulting store is organized and retrieved, and when external skills should be consolidated into slower parametric or shared stores. These choices are coupled: a wide edit repertoire requires stronger verification; a large store requires stronger routing and maintenance; and distillation is useful only when the fast-loop library is already selective enough to provide a clean training signal.

7.1 Edit repertoire: expansion, compression, and refactoring

The operator algebra of Equation 4 is most useful as a diagnostic for library maturity. The simplest systems are *expansion-only*: they add or refine skills after a task, as in Voyager, LATM, SAGE, ERL, and RetroAgent (Wang et al., 2023; Cai et al., 2023; Wang et al., 2025; Allard et al., 2026; Zhang et al., 2026e). This is enough for short horizons, but it makes the library monotone: every weak abstraction, duplicate, or stale instruction remains available to retrieval.

Mature systems add a second class of operators that compress or discipline the store. Prune, Merge, and Rerank appear in Wild-Skills, SkillRouter, AutoRefine, Trace2Skill, and AgentSkillOS; the recurring lesson is that a dynamic library needs a negative operator once distractor load matters. Growth

alone is not learning. The mechanism-level question is therefore not only how a system writes skills, but how it removes, merges, or demotes them when later evidence says they are unhelpful.

A third class changes structure rather than content. PSN applies rollback-gated refactors over an invocation graph; CODE-SHARP mutates a prerequisite DAG; SkillX splits hierarchies; CoEvoSkills abstracts multi-file `SKILL.md` packages; Bilevel-MCTS and SkillMOO optimize package structure and pass-rate/cost trade-offs (Shi et al., 2026; Bornemann et al., 2026; Wang et al., 2026a; Zhang et al., 2026c; Huang et al., 2026a; Gong et al., 2026). Structural edits are powerful because they can change reuse pathways, not just local skill text, but they also require lineage $\prec$ and rollback because a bad rewrite can invalidate many downstream invocations.

Parametric methods occupy a different regime. Their visible operator set often collapses to $\{\text{Distill}, \text{Abstract}\}$: Refine becomes more training, Merge becomes weight-space interpolation, and Prune is no longer a surgical library edit. This is why SkillsCrafter, MetaClaw, SKILL0, and K2-Agent are best read as two-timescale systems rather than ordinary library editors (Wang et al., 2026d; Xia et al., 2026b; Lu et al., 2026; Wu et al., 2026b).

7.2 Admission and verification are the selection mechanism

Verification is not an auxiliary module; it is the selection pressure that decides which generated artifacts become library state. Table 4 organizes the main verifier forms. Execution verifiers catch runnable failures through tests, contracts, simulators, or environment rollouts (SkillWeaver, EvoSkill, LIVE-SWE, SkillFoundry, SkillCraft). Judge-LLM verifiers assess semantic quality, pairwise preference, failure diagnosis, or release readiness (Trace2Skill, CoEvoSkills, AutoRefine, AgentSkillOS, SkillForge, MedSkillAudit). Rollback and audited-graph verifiers turn verification into a state-transition check, as in PSN and ASG-SI. Utility-based verifiers defer part of the decision to downstream use, as in Wild-Skills and EffiSkill.

The important distinction is what each verifier can observe. 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, but only on the probe set. Utility gates are cheap and deployment-realistic, but they may admit harmful or misleading skills before enough evidence accumulates. These limitations explain why high-capacity systems increasingly use staged admission: a candidate can be tentative, quarantined, promoted, demoted, or rolled back rather than simply accepted or rejected.

7.3 Storage and retrieval control the scaling regime

After admission, the bottleneck shifts from writing good skills to finding the right skills without importing distractors. Flat retrieval is attractive because it is simple, but the surveyed scaling studies repeatedly show a moderate-library-size drop. Single-Agent-Skills reports a sharp decline beyond roughly 64–128 skills when many agent skills are compiled into a single-agent library; Wild-Skills shows that realistic distractors can erase apparent gains from curated skills; and SkillRouter shows that 80K-scale registries require full-text retrieve-and-rerank rather than metadata-only selection (Li, 2026; Liu et al., 2026d; Zheng et al., 2026).

The response is to make storage reflect reusable structure. Hierarchies and capability trees support audit and specialization (AgentSkillOS, SkillX, Corpus2Skill); DAGs encode prerequisites or typed composition (CODE-SHARP, GraSP); invocation and relation graphs support refactoring, dependency-aware retrieval, and provenance (PSN, Graph of Skills, SkillGraph, ASG-SI) (Nie et al., 2026); ontologies and registries support portability and package-level governance (SkillNet, Skilldex). The mechanism-level insight is that retrieval quality is partly an indexing problem and partly a maintenance problem. Once a store grows, storage topology, reranking, pruning, and provenance become one system.

Form	Timing[a]	Admission[b]	Cost	Coverage	Representative methods
Execution tests / rollouts	W	hard	high	narrow[c]	Voyager, LATM, SkillWeaver, EvoSkill, LIVE-SWE, SkillCraft, SkillFoundry, Metasurface, MACRO
Judge-LLM rubric / meta-agent	W + M	hard / Pareto	medium	medium	Trace2Skill, CoEvoSkills, AutoRefine, AgentSkillOS, SkillNet, SkillOrchestra, CODE-SHARP, AgentFactory, SkillForge, MedSkillAudit[d]
Ensemble voting	W + M	maturity-gated	medium	wide	Agentic-Proposing[e]
Symbolic / audited	W	graph integrity	medium	wide	ASG-SI
Rollback validation	W + M	Δ-success $\leq 20\%$	medium	behavioural	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

[a] Timing codes: **W** = write-time, **I** = invocation-time, **M** = maintenance-time (periodic sweep over the library).
[b] Admission policies: *hard* thresholding = admit iff verifier returns success; *Pareto* = admit iff candidate dominates existing skills on a vector of criteria; *maturity-gated* = candidate enters as trial, promoted after usage budget; *graph integrity* = admit iff graph-level audit predicates hold (ASG-SI); Δ-*success* $\leq 20\%$ = tentative admission reverted if task-success rate on recent tasks drops by more than 20% (PSN); *stability-gated* = admit iff fast-loop performance variance is below threshold; *utility threshold* = defer admission, evict below utility floor.
[c] Execution verification is "narrow" because a passing unit test or rollout does not imply generalisation; this is partially why execution-verified methods cap their operator repertoire at {Add, Refine, Prune} (§7.2).
[d] AgentFactory's verifier is a meta-agent inspection of proposed subagents against a design specification; we classify it here as a judge-LLM variant (rather than as execution) because it inspects the proposal rather than running it against unit-test-style contracts.
[e] Agentic-Proposing uses a three-LLM-judge ensemble with majority voting. Agent0 and ASG-SI are driven by curriculum-reward and audit-gated reward respectively; they are not verifier ensembles in the sense used in this row and therefore do not appear here.

Table 4: **Verification architectures in the surveyed dynamic-skill methods.** Three axes organize the space: verifier *form*, verification *timing*, and *admission policy*. The two rightmost columns record the qualitative cost per candidate skill and the coverage over the class of defects each form can catch.

7.4 Update clocks separate adaptation from consolidation

Many recent systems separate a fast external loop from a slow internal loop. The fast loop edits files, procedures, code snippets, or graph nodes after tasks; the slow loop distills selected behavior into adapters, weights, shared libraries, or compact package formats. This separation lets the agent learn quickly without paying the cost or risk of continuous model updates, while still allowing stable skills to become cheaper to invoke later.

Table 5 compares the recurring decouplings. Some systems share the artifact between loops (MetaClaw, K2-Agent); others share reward or critique signals (SAGE, AgentEvolver, Tool-R0, COS-PLAY); others share curricula or verifiers (Agent0, SCALAR, ASG-SI). The key design variable is promotion: prevalence is cheap, reward is task-grounded, coverage favors behavioral diversity, and stability is closest to measuring whether distillation will preserve rather than corrupt a skill.

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. Conversely, 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.

Method	Mode[a]	Fast artifact	Slow artifact	Promoted on	Slow trigger	Slow vs. fast
K2-Agent (Wu et al., 2026b)	SA	SKILL.md + case	LoRA / adapter	reward ∩ prevalence	task end	augments
MetaClaw (Xia et al., 2026b)	SA	SKILL.md	LoRA	prevalence	periodic	augments
AgentEvolver (Zhai et al., 2025)	SS	NL heuristic	policy weights	RL replay buffer	RL update	augments
Tool-R0 (Acikgoz et al., 2026)	SS	tool proposals	policy weights	reward (dual self-play)	RL step	augments
SAGE (Wang et al., 2025)	SS	Python skill fn.	policy weights	skill-integrated reward	task end	augments[b]
COS-PLAY (Wu et al., 2026a)	SS	skill-bank entries	GRPO adapters	rollout reward + skill utility	co-evolution step	augments
Agent0 (Xia et al., 2025b)	SC	code proposals	executor weights	coverage / curriculum	co-evolution step	augments
SCALAR (Zabounidis et al., 2026)	SC	env / skill curation	policy weights	teacher signal	RL step	augments
ASG-SI (Huang & Huang, 2025)	SV	audited skill graph	policy weights	audit-gated reward	RL step	augments
Memento (Zhou et al., 2026)	—[c]	SKILL.md + case	(no slow loop)	—	—	fast-only
LSE (Chen et al., 2026d)	—[d]	prompt context	prompt + weight	reward	—	fast-only
Co-Evolving (Jung et al., 2025)	—[e]	code + critique	policy weights	hard-negative pool	batch complete	single-loop
Agentic-Proposing (Jiao et al., 2026)	—[f]	skill proposals	policy weights	ensemble-verifier pass	RL step	single-loop

[a] Decoupling mode (§7.4): **SA** = shared-artifact (fast and slow loops edit and distill the same textual skill); **SS** = shared-signal (both loops consume the same reward / critique stream at different cadences); **SC** = shared-curriculum (fast loop generates tasks for the slow loop); **SV** = shared-verifier (same verifier gates both loops at different stringency).
[b] SAGE's fast artifact is an executable Python skill function updated under a skill-integrated GRPO reward; the slow loop augments this artifact rather than replacing it, and we flag the row because it is the only **SS** method whose fast-loop artifact is code rather than a natural-language heuristic.
[c] Memento is listed as a fast-only baseline for contrast; it has no slow loop and therefore no decoupling mode applies.
[d] LSE trains a 4B edit policy that emits context-level edits as a learned action; there is no separate fast/slow decomposition because the learned editor *is* the fast loop and its weights are the only parametric artifact, so the four modes do not apply.
[e] Co-Evolving runs alternating DPO over a hard-negative trajectory pool; critiques and policy updates live in a single optimization loop, with no separate slow-loop distillation over a persistent skill library.
[f] Agentic-Proposing pairs a fast loop that proposes *problems* (not library edits) with an RL loop that updates the policy against verifier-gated rewards; because the fast-loop output is not a skill artifact, the two-timescale decoupling modes above do not apply.

Table 5: **Two-timescale decoupling modes in methods with both a fast in-context library and a slow parametric loop.** Rows are keyed by method and grouped by the four decoupling modes the literature converges on. The "Promoted on", "Slow trigger", and "Slow vs. fast" columns characterize what moves between loops, when, and whether the slow-loop output replaces or augments the fast-loop library.

7.5 Mechanism-level synthesis

The mechanism picture is compact. Dynamic skill systems improve when expansion is paired with compression, admission is paired with re-verification, retrieval is paired with structure, and fast editing is paired with slower consolidation. The same four requirements explain why apparently different systems occupy coherent regimes: executable-skill systems emphasize execution gates and small edit repertoires; natural-language lesson systems emphasize cheap proposal and later maintenance; graph and hierarchy systems trade storage complexity for retrieval and refactoring; and parametric systems trade editability for amortized inference. The strongest systems are not those with the largest libraries, but those that make the library easier to verify, route, repair, and consolidate over time.

8 Evaluation

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. This section audits evaluation through a lifecycle lens: how libraries are created, repaired, routed, compacted, and governed over time.

8.1 The Benchmark Landscape

The surveyed papers report on four partly-overlapping benchmark clusters. The first is the *agent-task* cluster: WebArena, VisualWebArena, Mind2Web, AgentBench, ST-Bench, SWE-bench-style command-line environments, and related terminal or software-engineering suites. These benchmarks evaluate an agent end-to-end on held-out tasks; dynamic-skill papers often inherit the benchmark and report a single success rate. The second is the *code-execution* cluster: HumanEval, MBPP, LiveCodeBench, APPS, and software-engineering task suites with intrinsic execution verification. These benchmarks are why code-skill and skill-aware RL papers can afford stronger admission gates. The third is the *reasoning* cluster: AIME, MATH, GPQA, and HLE; this is where Agentic-Proposing and Memento report some headline results, and where verifier quality is often load-bearing because ground truth is unambiguous.

The fourth cluster is most specific to this survey: *skill-lifecycle evaluation*. SkillsBench (Li et al., 2026c) measures skill usefulness across tasks, Wild-Skills (Liu et al., 2026d) stresses retrieval realism under distractors and 34K-candidate retrieval, Single-Agent-Skills (Li, 2026) gives the clearest controlled size sweep for the 64–128-skill flat-retrieval knee, and SkillRouter (Zheng et al., 2026) evaluates full-text routing over an 80K skill pool. SkillFlow-Bench (Zhang et al., 2026h) is the most lifecycle-aligned benchmark: 166 runnable tasks across 20 DAEF-structured families, empty initial family libraries, JSON skill patches from trajectories, and reports of completion, turns, cost, output tokens, final skill count, and skill-use rate. Its Table 1 gives the key finding that skill use is not skill utility: Claude Opus 4.6 improves from 62.65% to 71.08%, while Kimi K2.5 gains only +0.60 points despite 66.87% skill use and GPT 5.3 Codex regresses by 6.02 points.

SkillLearnBench (Zhong et al., 2026) asks whether agents can continually generate useful skills, not merely use supplied ones, and finds a large gap to human performance across 20 verified skill-dependent tasks. HarmfulSkillBench (Jiang et al., 2026c) is the safety analogue, separating harmful skill presence from explicit and implicit invocation. MedSkillAudit (Hou et al., 2026) adds a domain-release protocol: evaluate the reusable skill artifact itself for release readiness.

8.2 Reported Metric Families

Four metric families dominate. *Terminal success rate* or pass@k remains the default: evaluate after training or after library construction and report a single number. This metric is useful but incomplete because it hides library-size knees, focus effects, and maintenance effects. *Sample efficiency* or wall-clock-to-threshold metrics (EvoSkill, SAGE, K2-Agent) report how quickly a method reaches a target success rate; these are better for surfacing the weaker-backbone advantage that dynamic skills often show. *Library-size and retrieval stress tests* (Single-Agent-Skills, Wild-Skills, SkillRouter, Graph of Skills) expose scaling behavior,

Benchmark	Primary lifecycle coverage	Self-gen.	Revision	Trajectory	Usage utility	Main limitation for this survey
SkillsBench	skill usefulness under provided or generated skills	✓	–	–	–	evaluates skill effectiveness more than longitudinal library management
Wild-Skills	realistic retrieval, distractors, curation utility over a 34K pool	–	–	∼	✓	strong retrieval realism, but not sequential self-repair
SkillRouter	full-text routing at scale over an 80K skill pool	–	–	–	∼	measures retrieval accuracy/speed more than downstream lifecycle utility
Graph of Skills	dependency-aware retrieval over 200–2,000 skill libraries	–	–	–	✓	strong structural retrieval evidence, but no skill repair loop
LifelongAgentBench	lifelong task sequences and accumulation pressure	∼	∼	✓	–	adjacent lifelong-learning benchmark, not skill-artifact-specific
ProEvolve	programmable distribution shift for evolving agent benchmarks	–	–	✓	–	supplies drift protocol but not a skill-library protocol
SkillFlow-Bench	discovery, JSON skill patches, repair, reuse, and compactness across 166 tasks	✓	✓	✓	✓	family reset avoids global heterogeneous-library retrieval; model and harness are co-varied
SkillLearnBench	continual skill generation on 20 verified skill-dependent tasks	✓	✓	✓	✓	evaluates generation quality, but task count is still small
ClawSafety	safety surfaces for skill injection in personal agents	–	–	–	✓	attack benchmark, not a defended dynamic-skill evaluation
HarmfulSkillBench	harmful skill prevalence and explicit/implicit unsafe invocation	–	–	–	✓	safety benchmark; not a repair or mitigation protocol
MedSkillAudit	domain-specific release-readiness audit for medical research skills	–	✓	–	∼	reliability study over 75 skills, not downstream task benchmark

Table 6: **Benchmark coverage over dynamic-skill lifecycle dimensions.** The table separates benchmark roles rather than ranking benchmarks. "Self-gen." means the benchmark evaluates skills produced by the agent; "Revision" means skills can be patched or repaired over time; "Trajectory" means the protocol exposes a time-ordered library or task sequence; "Usage utility" means the benchmark can distinguish reading/calling a skill from actually improving task outcome.

but only a minority of papers report them. *Lifecycle metrics*, newly visible in SkillFlow-Bench and SkillLearnBench, report final skill count, file-kind composition, skill-use rate, generated-skill quality, and the gap between skill usage and task improvement.

XSkill (Jiang et al., 2026a) adds a multimodal metric design: Average@4 and Pass@4 over repeated rollouts, plus ablations over the skill stream, experience stream, and knowledge managers. It is not a full library-trajectory protocol, but it separates average rollout quality from best-of-four exploration and shows that transfer can raise Pass@4 while lowering Average@4 on weaker open-source backbones.

Two quantities remain under-reported. The first is *operator velocity*: counts of Add, Refine, Merge, Prune, and Rerank per task or per dollar. Without this quantity, the velocity–soundness tradeoff between PSN, CODE-SHARP, SkillMOO, and Bilevel-MCTS cannot be compared quantitatively. The second is *repair quality*: whether a later skill patch actually corrects a faulty abstraction. SkillFlow-Bench exposes this qualitatively through incorrect-skill drift and skill inflation, and SkillForge does so in a deployment-style failure-diagnosis loop, but the field still lacks a standard scalar repair metric.

8.3 Four Comparisons That Remain 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. *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: adjacent protocols such as ELL/StuLife (Cai et al., 2025), LifelongAgentBench (Zheng et al., 2025b), and ProEvolve (Li et al., 2026a) expose lifelong, proactive, or programmable shift, but most deployment papers still use bespoke task streams. These limits determine how the next section treats evidence: the regularities in §9 rely mainly on within-paper ablations and convergent benchmark behavior rather than cross-paper leaderboards.

8.4 Toward Trajectory-Aware Evaluation

A trajectory-aware protocol should report the library as a time series. Four elements are sufficient: performance, skill count, and retrieval quality at a grid of task indices or library sizes; operator velocities for Add, Refine, Merge, and Prune; a drift schedule or family-transfer condition; and a maintenance-off or repair-off ablation. Single-Agent-Skills' size sweep, SkillRouter's 80K routing benchmark, and SkillFlow-Bench's skill-count trajectories are complementary starting points.

SkillFlow-Bench moves the field in this direction, but its family-reset design leaves open how one global library behaves under heterogeneous workflows. The next step is to add cross-family global-library protocols, operator-velocity logging, and maintenance-off ablations.

9 Seven Empirical Regularities

Building on the evaluation audit in §8, the primary dynamic-skill cluster within the 94-paper modern audit set surfaces seven empirical observations that recur across methods, benchmarks, and artifact types. We state each as a *regularity*, not a theorem: the qualitative effects are consistent, but effect sizes and boundary conditions vary. Evidence is strongest for within-paper ablations, moderate for convergent benchmark behavior, and weakest for architectural corroboration without ablation.

Table 7 summarizes the evidence for the section and assigns each regularity the evidence grade defined in §2.5; § 12 uses these regularities to frame the open-problem agenda.

9.1 Curated skills outperform unverified self-generated skills

One of the most consistently replicated findings in the surveyed corpus is that *admission matters*. Libraries whose write-time verifier is stronger than "the agent proposed it" consistently beat libraries that admit any proposal, and the evidence spans methods at both ends of the verification spectrum. EvoSkill (Alzubi et al., 2026) reports that its Pareto-front selection over held-out validation performance avoids the redundant/conflicting-skill accumulation seen under greedy acceptance and yields +7.3 absolute points on OfficeQA (Table 1 / Figure 2) and +12.1 points on SealQA; CoEvoSkills (Zhang et al., 2026c) provides the cleanest verifier ablation in the corpus, with Table B1 showing that removal of the surrogate verifier drops SkillsBench pass rate from 71.1% to 41.1%; SkillWeaver (Zheng et al., 2025a) reports that removing the "practice + verify" phases of PPVH (leaving only propose+hone) drops benchmark performance below the no-skill-library baseline on its hardest web tasks; and Trace2Skill (Ni et al., 2026) reports that prevalence-weighted consolidation is useful only when consolidation also filters on a judge score. The April 2026 systems strengthen the same point from new domains: SkillFoundry (Shen et al., 2026) only admits scientific packages after contract/provenance/test validation, and SkillForge (Liu et al., 2026c) improves deployed support skills through failure analysis, diagnosis, and optimization rather than blind rewriting. The regularity also appears in maintenance-heavy systems: AutoRefine (Qiu et al., 2026)'s pruning/merging gate and AgentSkillOS (Li et al., 2026b)'s audit-gated organization both report that disabling the gate erodes downstream task success.

One-line claim	Grade	Primary supporting papers	Key caveat / exception class
Admission gates matter: curated skills beat unverified self-generated skills.	A	EvoSkill, CoEvoSkills, SkillWeaver, Trace2Skill, SkillFoundry, SkillForge, AutoRefine, AgentSkillOS	Case-based memory (SimpleMem, parts of MUSE) where admission = retrieval relevance; hard admission filters can remove rare-task cases.
In skill-aware RL, verifier *quality* is often one of the most decisive engineering choices.	A	CoEvoSkills, CODE-SHARP, Agentic-Proposing, Co-Evolving, SELAUR	ASG-SI is architectural corroboration, not benchmark-strength ablation evidence; executable-library regimes with strong execution verifiers can see diminishing returns.
Flat retrieval often drops at moderate library scale (roughly one hundred skills).	B	Single-Agent-Skills, Wild-Skills, AgentSkillOS, SkillRouter, Graph of Skills	Single-Agent-Skills supplies the controlled 64–128-skill curve; SkillRouter supplies 80K-scale routing evidence rather than a smooth size sweep.
Several studies report larger relative gains for weaker backbones.	B	SkillWeaver, MetaClaw, EvoSkill, Agentic-Proposing	Libraries of rare-task specializations can invert the effect: strong models route reliably to them while weaker models cannot; XSkill shows transferred knowledge can improve Pass@4 while hurting Average@4 on weaker backbones.
Focused libraries often beat comprehensive ones even when the focused one is a subset.	B	SkillX, Wild-Skills, CASCADE, SkillMOO, SkillLearnBench, SkillFlow-Bench, SKILL.md-trimming literature	Non-stationary deployment (SkillClaw, AutoSkill, MetaClaw): a focused library staleness-dominates; the result becomes an argument for fast Prune.
At moderate-to-large library sizes, maintenance becomes load-bearing.	A/B	AutoRefine (TravelPlanner maintenance ablation: $35.6 \rightarrow 31.1$, $4.5\times$ repository growth, $0.71 \rightarrow 0.08$ utilization), AgentSkillOS, Wild-Skills, EffiSkill, PSN, SkillFlow-Bench, MetaClaw, SKILL0	Very short task horizons (a handful of tasks pre-evaluation) where maintenance cost cannot amortize; SkillFlow-Bench is benchmark corroboration, not a maintenance-off ablation.
Write-time abstraction (Abstract/Distill at authoring) is usually the stronger backbone than read-time abstraction alone.	B/C	Trace2Skill, CASCADE, SimpleMem, XSkill, SkillFlow-Bench	SimpleMem and XSkill are adjacent memory / dual-store evidence; SkillFlow-Bench is a structured-externalization control rather than a clean write/read ablation; non-stationary deployments can require read-time adaptation layered on top.

Table 7: **Seven empirical regularities that replicate across the dynamic-skills literature (2023–2026).** Each row summarizes one stylized fact (§9), an evidence grade using the convention in §2.5, the methods that provide supporting ablations or measurements, and the known exception class. The strongest themes concern write-time discipline; the retrieval-scaling and focus rows argue for smaller libraries from the retrieval-resolution and distractor-load ends respectively; the weaker-backbone row is the most deployment-facing. The open-problem agenda in §12 is organized around these seven regularities.

Caveats. This regularity says that replacing no verification with some verification is usually valuable, not that more verification is always better. The main exception is the memory/trajectory family (SimpleMem, parts of MUSE) (Yang et al., 2025a), where artifacts are episodic cases and admission is closer to relevance filtering than quality control.

9.2 Verification *quality* is often decisive in skill-aware RL

Inside skill-aware RL, the *quality of the verifier or reward-shaping signal* is often one of the most load-bearing choices. CoEvoSkills' Table B1 surrogate-verifier ablation isolates a 30-point verifier effect; CODE-SHARP improves success from 24.30% to 41.02% through refinement mutations under a learned gate; Agentic-Proposing's verifier ensemble plus dynamic pruning improves problem validity from 68.7% to 82.3%; Co-Evolving (Jung et al., 2025) attributes substantial gain to hard-negative construction; and SELAUR (Zhang et al., 2026b) shows that uncertainty-aware reward shaping can matter as much as the RL algorithm. ASG-SI (Huang & Huang, 2025) is architectural corroboration: it specifies the audited graph and verifiable-reward machinery a deployed system would need.

Caveats. This regularity is specific to skill-aware RL. In executable libraries, once execution verification exists, additional rollouts or tighter contracts may have diminishing returns relative to Prune and Rerank.

9.3 Flat retrieval often drops around 64–128 skills

Flat skill libraries often show an accuracy drop around one hundred skills, consistent with a top-k retrieval signal-to-noise collapse. Single-Agent-Skills (Li, 2026) gives the clearest controlled size sweep: selection remains high at 16–64 skills, degrades around 128 skills, and drops sharply at 256 skills unless hierarchical routing or disambiguating instructions are added. AgentSkillOS (Li et al., 2026b) motivates its capability tree with flat-invocation collapse, Wild-Skills (Liu et al., 2026d) shows the same mechanism under forced loading, autonomous selection, distractors, 34K-pool retrieval, and no curated task-specific skills, and SkillRouter (Zheng et al., 2026) shows that 80K-scale registries need full-text retrieve-and-rerank because metadata-only routing loses implementation-level signals. Graph of Skills (Liu et al., 2026a) adds 200–2,000-skill evidence that dependency-aware graph retrieval can preserve reward while compressing token use.

Caveats. The knee's location depends on retriever quality, description length, skill orthogonality, and whether the corpus exposes full skill bodies. Hierarchical, DAG, and ontology storage can push it rightward (AgentSkillOS, SkillX, SkillOrchestra) (Wang et al., 2026b); full-text reranking (SkillRouter) addresses the complementary registry-scale routing problem (Zheng et al., 2026). The literature has not shown general removal.

9.4 Several studies report larger relative gains for weaker backbones

Several studies report larger relative gains for weaker base models than for stronger ones, but the evidence is convergent rather than a controlled cross-paper law. SkillWeaver (Zheng et al., 2025a) closes a larger fraction of the capability gap on weaker web-agent backbones; MetaClaw (Xia et al., 2026b) reports larger relative two-timescale gains on weaker backbones in its setting; EvoSkill reports analogous gap compression; and Agentic-Proposing's headline result is on a 30B model, with a smaller relative lift on the frontier teacher. Because these papers vary in harness, context budget, and gain definition, we treat the pattern as a deployment-facing regularity rather than as a comparable effect-size estimate.

Caveats. The effect depends on library content. Common-but-not-obvious heuristics help weaker models more; rare specializations can invert the pattern if weaker models cannot route to them reliably. XSkill's Qwen transfer results are the caution: transfer can raise Pass@4 by encouraging exploration while lowering Average@4.

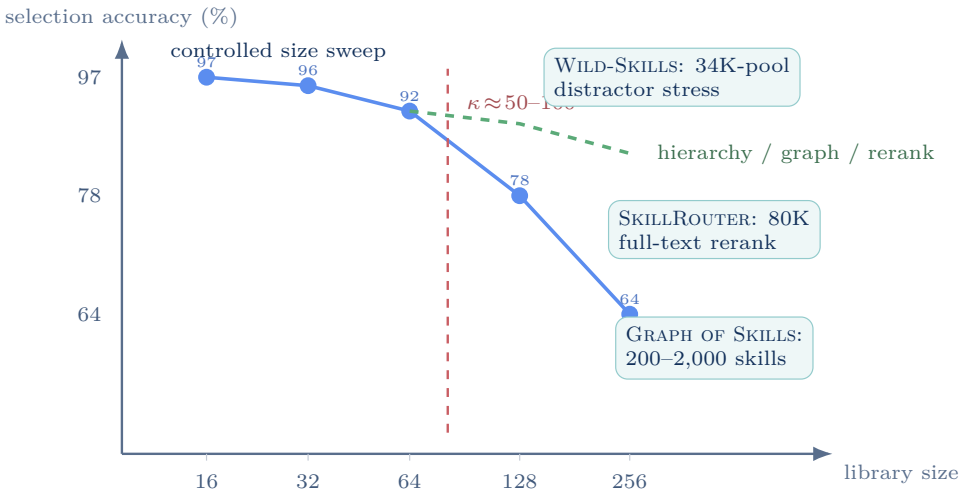

Figure 4: **Retrieval-scaling evidence behind R3.** The blue curve plots the controlled SINGLE-AGENT-SKILLS sweep reported in the paper: 16–32 skills remain around 96–98%, 64 skills at 92%, 128 skills at 78%, and 256 skills at 64%. The dashed green path summarizes the mitigation family rather than a pooled leaderboard: hierarchy, graph retrieval, and full-text reranking push the failure rightward in SINGLE-AGENT-SKILLS, WILD-SKILLS, SKILLROUTER, and GRAPH OF SKILLS, but they are not yet comparable under one shared harness.

9.5 Focused libraries often beat comprehensive ones

A library curated to a narrow task distribution can outperform a more comprehensive library, even when the former is a subset of the latter. This is the content-level dual of the retrieval knee: the failure mechanism is distractor load. SKILLX (Wang et al., 2026a) reports that task-family-tuned hierarchies beat flat libraries; WILD-SKILLS shows that retrieval-utility filtering beats skills that looked useful in isolation; and CASCADE (Huang et al., 2025) reports that consolidation beats pure growth at fixed compute. SKILL-MOO (Gong et al., 2026) optimizes pass rate and cost jointly, SKILLLEARNBENCH (Zhong et al., 2026) shows that plausible skill text does not imply competence, and SKILLFLOW-BENCH (Zhang et al., 2026h) finds that stronger settings maintain compact revised skills while weaker settings often fragment into many low-utility files. The single-skill analogue is the SKILL.md focused-prompt result: trimming L2 prose to task-relevant instructions improves use.

Caveats. In non-stationary deployments (SKILLCLAW, AUTOSKILL, METACLAW), focused libraries can become stale; the result then argues for fast PRUNE, not permanent narrowness.

9.6 At moderate-to-large library sizes, maintenance becomes load-bearing

Systems with at least one negative operator—PRUNE, MERGE, or an equivalent—often beat monotonic-growth systems once distractor load matters. AUTOREFINE (Qiu et al., 2026) reports the headline ALFWorld result in its Table 1, but its maintenance-off evidence is the TravelPlanner validation ablation: removing periodic pruning and merging lowers final pass rate from 35.6% to 31.1%, grows the repository by 4.5×, and reduces utilization from 0.71 to 0.08 (its Figures 2–3). AGENTSKILLOS attributes scaling to periodic capability-tree audit; WILD-SKILLS and EFFISKILL (Wang et al., 2026f) find utility-based filtering can beat increased admission; and PSN's maturity gating loses its stated advantage when disabled. SKILLFLOW-BENCH adds that weaker settings often fail by skill inflation and incorrect-skill drift rather than inability to write skills. Parametric systems express the same point through distillation windows (METACLAW, SKILL0) (Lu et al., 2026): mistimed consolidation lets the fast-loop library grow without bound.

Caveats. The exception is very short horizons, where maintenance cost has no time to amortize.

9.7 Write-time abstraction is usually the stronger backbone than read-time abstraction alone

The literature usually favors *write-time* abstraction—retrospective induction, Abstract, or Distill during skill authoring—over read-time summarization alone. Trace2Skill (Ni et al., 2026)'s prevalence-weighted consolidation beats a read-time summarizer baseline; CASCADE's write-time distillation beats a retrieval-plus-rerank control; and SimpleMem (Liu et al., 2026b) shows in Table 5 that removing write-time semantic compression drops LoCoMo average F1 from 43.24 to 31.29. XSkill (Jiang et al., 2026a) extends this finding to multimodal agents: its Table 3 ablation shows that removing the Experience Manager drops VisualToolBench Average@4 by 4.09 points and removing the Skill Manager drops it by 3.62, while read-time task-decomposition/adaptation ablations are smaller. SkillFlow-Bench supports file-level write-time abstraction but is not a clean write-time-versus-read-time ablation; K2-Agent (Wu et al., 2026b)'s declarative-procedural split is architectural corroboration.

Caveats. The result is cleanest under reasonably stationary tasks. In non-stationary deployments and visually grounded multimodal settings, read-time adaptation layered on top of write-time abstraction can still be necessary.

9.8 Cross-regularity observations

Three observations connect the regularities. Admission, verifier quality under RL, maintenance, and write-time abstraction are all forms of *write-time discipline*. Retrieval scaling and focused-library effects both argue that smaller can be better, through different mechanisms: retrieval resolution and distractor load. Finally, lifecycle benchmarks warn that skill *usage* is not skill *utility*; Wild-Skills shows this through retrieval realism, and SkillFlow-Bench through high-use, low-gain settings. The open problems in §12 either exploit these regularities or ask why a method appears to violate one without visible cost.

10 Infrastructure

Dynamic-skill methods depend on packaging formats, storage backends, marketplaces, SDKs, and edit-execution pipelines. These choices were peripheral in 2023 but are load-bearing by 2026: they define the skill unit, determine scaling limits, shape cross-user aggregation, and decide which operators can be implemented at useful velocity.

Table 8 groups representative systems by structural constraints, not quality. The visible diagonals—flat storage with near-monotonic {Add}, hierarchies with Split-heavy maintenance, and DAG/ontology stores with Compose-heavy workflows—explain why methods rarely transfer cleanly across infrastructure stacks: off-diagonal operators are often too expensive to run at the required edit velocity.

10.1 Packaging formats and the minimum viable unit

The surveyed literature uses four packaging formats. `SKILL.md` packages dominate the agent-skill regime: one directory per skill, with a descriptor, optional scripts, and an interface manifest. Function-calling schemas and MCP/plugin manifests dominate tool-as-skill systems, where the portable object is a typed callable signature. Hybrid skill-memory stores, as in XSkill (Jiang et al., 2026a), pair a Markdown workflow with a JSON-like experience bank. A fourth, less explicit format treats trajectories themselves as retrievable artifacts (SimpleMem, MUSE). Recent infrastructure papers add package-manager and corpus-compiler variants: Skilldex (Saha & Hemanth, 2026) gives skills scoped package semantics, while Corpus2Skill (Sun et al., 2026) compiles an enterprise corpus into a navigable skill directory.

The formats trade portability against structure. `SKILL.md` travels across backbones but depends on the receiver's ability to follow prose; function schemas travel across models but assume the target tool exists; hybrid stores require both workflow and experience interfaces; trajectory memories grow freely but are task-specific. K2-Agent's declarative-procedural pair and XSkill's Markdown-plus-experience split are the closest attempts to bridge formats, but both still assume compatible tools and context interfaces.

Dim.	Class	Representative systems	Fast ops[a]	Costly ops[a]	Primary trade-off
Package	SKILL.md	SkillClaw, AutoSkill, MetaClaw, Skilldex	Add, Refine	Compose	portability/lock-in
	Tool schema	Function-calling + MCP/plugin manifests	Add, Compose	Abstract	schema/capability
	Hybrid skill/memory	XSkill, Memento, SAGER[b]	Add, Rerank	Merge, Prune	portability/ compat.
	Compiled corpus	Corpus2Skill, SkillRepoMining, SkillFoundry	Add, Abstract	Prune	provenance/quality
	Trajectory	SimpleMem, MUSE[b]	Add, Rerank	Abstract, Distill	read-time/write-time
Storage	Flat embedding	EvoSkill, SkillWeaver, Agentic-Proposing	Add	Merge	admit/reorganise
	Hierarchical tree	AgentSkillOS, SkillX, Corpus2Skill	Split	Prune	restructure/remove
	Graph / dependency	PSN, CODE-SHARP, Graph of Skills, GraSP	Compose[c]	Prune	chain/remove
	Typed ontology	SkillOrchestra	Compose (typed)	Refine	expressivity/cost
Market	Pull-model	Central curated registry; Skilldex	Add, Rerank	Merge	curation/scale
	Push-model	SkillClaw cross-user	Add	Merge, Prune	velocity/ provenance
	Hybrid	AutoSkill dual review	Add, Rerank	Merge, Prune	review/throughput
	Security scanner	SkillSieve, Malicious-or-Not, Credential Leakage	Prune	Refine	recall/false positives
Pipeline	Inline edit	PSN, CODE-SHARP	Refine, Rewrite	Prune	velocity/rollback
	Log-and-apply	AutoRefine, AgentSkillOS	Prune	Refine	audit/latency
	Shadow exec.	MetaClaw, SKILL0	Distill	Refine	safety/staleness
	Release audit	MedSkillAudit, AgentDevel	Prune	Refine	rigor/throughput

[a] "Fast" / "costly" are relative within a class: listed entries are drawn from the ten-operator algebra (§7.1) and flag which operators a given infrastructure stack admits cheaply (unit-cost edits) versus which require a heavier mechanism (locking, re-indexing, cross-user consensus, or human review).
[b] SimpleMem and MUSE are trajectory-memory systems in which episodes themselves play the skill-like role of retrievable artifacts; XSkill and Memento are hybrid cases that pair skill documents with case / experience stores. We include these methods when their artifact is read by an agent loop as if it were a skill. The central dynamic-skills-vs-memory distinction is preserved by the *Artif.* and *Trig.* columns of Table 11.
[c] For PSN and CODE-SHARP, the graph is cheap for library-edit-level Compose to *audit* (prerequisites are explicit); admitting a composed skill as a new library entry still requires a verifier pass, so the cheapness is structural, not free.

Table 8: **Four infrastructure dimensions (package, storage, market, pipeline) that materially constrain dynamic skill systems.** Each row classifies a representative deployment along the same categorical vocabulary so that the operator economy and the primary trade-off are readable at a glance. The "fast ops" / "costly ops" asymmetry is the core claim of the table: infrastructure choices predetermine which operator algebra is economical, which is why the operator–storage diagonal of the master taxonomy (Table 11) is sparsely populated.

10.2 Storage backends and scaling limits

Storage determines whether the moderate-library-size retrieval knee is conceded or delayed. Flat embedding indices (EvoSkill, SkillWeaver, Agentic-Proposing) use dense retrieval over descriptions and are the class most directly implicated by the scaling evidence. Hierarchical stores (AgentSkillOS, SkillX, Corpus2Skill) shift the knee; graph stores (CODE-SHARP, PSN, Graph of Skills, GraSP) expose skill–skill relations to retrievers and audits; ontology stores (SkillOrchestra) attach semantic categories for typed composition.

For dynamic systems, the key question is operator cost under the storage class. Flat indices make Add cheap and Merge expensive; hierarchies make Split cheap and subtree Prune expensive; DAGs make Compose easier to audit but Prune risky because descendants may depend on the removed node; ontologies make category inheritance cheap but category-changing Refine expensive. This operator–storage coupling explains why the master table's off-diagonal combinations remain sparse.

10.3 Marketplaces, plugins, and cross-user aggregation

A 2026 development is the skill *marketplace*: a registry where skills authored by one user or team are listed, reviewed, ranked, and adopted by others. Marketplaces change admission from a single verifier score to an aggregate utility and moderation signal. Agent Skills: Data-Driven Analysis (Ling et al., 2026) measures the speed of the public Claude-style skill ecosystem; Malicious-or-Not (Holzbauer et al., 2026) and Credential Leakage (Chen et al., 2026e) show why registry metadata, repository context, and remediation workflows are now infrastructure, not a separate security appendix.

The literature distinguishes pull-model registries with central admission, push-model sharing as in SkillClaw, hybrid registries with per-user overrides as in AutoSkill, and package-manager models such as Skilldex with scoped installs and conformance checks. Pull models foreground retrieval at scale; push, hybrid, and package-manager models foreground attribution and governance.

10.4 The edit–execute pipeline

Each operator in §7.1 must be implemented as an edit to a concrete artifact. Three pipeline architectures recur. *Inline editing* (PSN, CODE-SHARP) edits the artifact in place and needs an explicit version store for rollback. *Log-and-apply* pipelines (AutoRefine, AgentSkillOS) stage the edit, verify it, and commit only after admission. *Shadow-execution* pipelines (MetaClaw, SKILL0) evolve a shadow library while serving from the main one, then cut over during a distillation window.

These pipelines determine whether a proposed operator is cheap enough to use continuously or expensive enough to reserve for release gates.

11 Safety and Governance

The April 2026 safety wave turns dynamic skills from a prompt-injection concern into a software-supply-chain problem. The corpus now includes attack taxonomies, registry-scale measurements, admission scanners, credential-leakage studies, harmful-skill benchmarks, repository-context audits, and skill-stealing attacks. Because skill artifacts combine natural language, code, configuration, sometimes learned models, and social provenance, admission control, provenance, sandboxing, and release governance must be lifecycle stages.

Table 9 summarizes the evidence. The field has attack studies and partial defenses, but few defended lifecycle systems: methods verify utility but not maliciousness, scanners analyze artifacts but not downstream composition, and registries expose popularity but not operator-level lineage. Dynamic skill stores need a security pipeline that mirrors their lifecycle pipeline.

Surface	Most exposed lifecycle point	Current evidence	Required primitive	Coverage state
Skill-embedded prompt injection	admission / invocation	ClawSafety: skill files highest-ASR vector (69.4% avg.); SkillAttack: attack-path refinement over adversarial and real skills	adversarial SKILL.md sanitization + invocation-time guard	attack measured; weak defense
Supply-chain poisoning	publication / install	Supply-Chain-Poisoning: 1,070 adversarial skills, 11.6–33.5% bypass; BadSkill: 97.5–99.5% ASR for model-in-skill backdoors	package manifest, model provenance, sandboxed install	attack measured; partial scanners
Harmful but valid skills	deployment / policy gate	HarmfulSkillBench: 4,858 harmful of 98,440 collected skills; installed skills increase harm scores	capability-policy release gate	benchmarked; immature governance
Credential / secret leakage	execution / stdout / files	Credential Leakage: 520 affected of 17,022 sampled skills; 76.3% cross-modal NL+code cases; 73.5% debug-log vector	cross-modal secret scanner + stdout containment	measured; remediation partial
Scanner false positives	admission / registry review	SkillSieve: 0.800 F1, low-cost triage; Malicious-or-Not: repository context leaves 0.52% of scanner-flagged skill-repo pairs malicious	repository-aware triage and provenance scoring	partial defense + calibration
Skill theft / IP leakage	marketplace / black-box use	SkillStealing: proprietary skills can leak semantically through few interactions	ownership metadata, rate limits, watermarking, access control	measured; no mature defense
Domain release readiness	pre-deployment review	MedSkillAudit: 57.3% below Limited Release; system-expert ICC 0.449 on 75 medical skills	domain-specific audit rubric and release tiers	early domain audit
Attribution / composition misuse	maintenance / composition	ASG-SI audited graphs; SkillOrchestra typed routing; no full operator-level audit	operator-granular lineage + composition-time review	architectural only

Table 9: **Safety surfaces for dynamic skill systems after the April 2026 literature wave.** The table distinguishes attack evidence, defense evidence, and governance primitives. Skill safety is not a single prompt-injection benchmark: it includes supply-chain poisoning, harmful-but-valid capability packages, credential leakage, scanner calibration, skill theft, and domain release readiness.

11.1 Eight safety surfaces specific to dynamic skills

The surfaces below are specific to, or strongly amplified by, lifecycle-managed skill stores. Towards Secure Agent Skills (Li et al., 2026e) organizes threats across creation, distribution, deployment, and execution; we map that phase structure onto the survey's operators and artifact-store formalism.

Prompt injection through admitted skills. An admitted skill can carry adversarial instructions in documentation, examples, comments, or generated helper files. ClawSafety (Wei et al., 2026) quantifies the outcome-level risk: malicious skill files are the highest-ASR injection vector in its OpenClaw personal-agent benchmark, averaging 69.4% ASR across backbones. SkillAttack (Duan et al., 2026) shows the complementary red-team view: benign-looking skills can become exploitable when an attacker refines prompts into attack paths, with injected adversarial skills reaching high ASR and real-world Hot100 skills still exploitable under some conditions.

Supply-chain poisoning at publication and installation. Skill packages can be poisoned before a user ever invokes them. Supply-Chain-Poisoning (Qu et al., 2026) constructs DDIPE attacks over 1,070 adversarial skills across 15 MITRE ATT&CK categories and reports bypass rates from 11.6% to 33.5% depending on defense configuration, with 2.5% evading both static and alignment filters. BadSkill (Tie et al., 2026) adds a distinct model-in-skill threat: a skill can bundle a backdoored model whose malicious behavior is hidden in learned parameters, reaching 97.5–99.5% ASR while largely preserving benign accuracy.

Harmful but well-formed skills. A skill can be valid, useful, and non-poisoned while still enabling harmful functionality. HarmfulSkillBench (Jiang et al., 2026c) finds 4,858 harmful skills among 98,440 collected skills and shows that installed harmful skills can raise harm scores even under implicit invocation. This is a policy-governance problem rather than malware detection.

Credential and secret leakage. Credential leakage is a cross-modal property of skills, because secrets can be exposed only through the interaction of SKILL.md text, code, runtime stdout, and agent context. Credential Leakage (Chen et al., 2026e) samples 17,022 skills from a 170,226-skill SkillsMP population, identifies 520 affected skills and 1,708 issues, and reports that 76.3% of cases require joint NL+code analysis. Debug logging accounts for 73.5% of vulnerability issues because stdout is often fed back to the LLM, turning routine logs into credential disclosure.

Admission scanner limits and false positives. Security scanners are necessary but not sufficient. SkillSieve (Hou & Yang, 2026) is one of the strongest current defense-side results in this corpus, filtering 86% of skills at zero API cost and reporting 0.800 F1 at about $0.006 per skill. Malicious-or-Not (Holzbauer et al., 2026) shows the opposite failure mode: scanner-only classification can badly overstate risk, while repository context reduces 2,887 scanner-flagged skill-repository combinations to 0.52% remaining in malicious flagged repositories. Admission gates therefore need both content analysis and provenance/context analysis.

Ownership, confidentiality, and skill theft. Once skills become valuable artifacts, the market creates confidentiality and intellectual-property risks. SkillStealing (Wang et al., 2026e) studies black-box extraction from proprietary agents and reports that paid or proprietary skill behavior can be inferred with only a few interactions, with substantial semantic leakage even when exact text recovery is low. This surface is orthogonal to user harm: the victim may be a skill author or registry operator rather than the end user.

Domain-specific release readiness. High-stakes domains need release gates beyond generic maliciousness. MedSkillAudit (Hou et al., 2026) evaluates 75 medical research skills, finds that 57.3% fall below the Limited Release threshold, and reports automated-audit ICC(2,1)=0.449 against expert consensus. The result supports domain-specific rubrics for scientific integrity, reproducibility, and boundary safety.

Attribution loss and composition misuse. Maintenance operators can delete, merge, split, and rewrite skills. Without operator-level provenance, later audits cannot reconstruct which artifact contributed to a

decision. Composition adds another risk: individually acceptable skills can be harmful when chained. ASG-SI (Huang & Huang, 2025) is the closest surveyed system to graph-level audit, and SkillOrchestra (Wang et al., 2026b) gestures toward typed composition, but neither closes the full provenance-plus-composition safety loop.

11.2 Coverage gap across the surveyed literature

The table's main implication is that skill safety is a lifecycle property: the vulnerable point can be admission, publication, installation, execution, marketplace review, or later composition.

11.3 Governance primitives a deployment would need

A defensible deployment needs six primitives: an *admission-time scanner* combining static checks, LLM/rubric analysis, sandboxing, and repository provenance; a *package manifest* for dependencies, permissions, model artifacts, network endpoints, and allowed tools; a *runtime containment layer* for stdout, filesystem, network, and bundled-model 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, remediation, and abandoned-repository hijacking.

If a system can create, modify, distribute, retrieve, and compose skills automatically, it must also audit those operations automatically.

12 Open Problems

The seven regularities constrain the next research agenda: new methods should exploit them or explain why their setting violates them. This section states eight open problems, organized around admission, maintenance, retrieval/composition, and deployment. We label each problem's evidence basis as *measured* when it follows from direct benchmark or deployment evidence, *extrapolated* when it extends measured behavior beyond the tested regime, and *speculative* when the literature has not yet run the necessary horizon or scale. Table 10 gives a concrete first experiment for each.

12.1 Compositional verifiers for code skills

Evidence basis: extrapolated from measured verifier ablations. Verifier design is decisive in skill-aware RL, and PSN's rollback-gated behavioural validation and CODE-SHARP's learned judges expose a useful tradeoff: probe-set behavioral coverage versus broader but less grounded judge coverage. No surveyed method composes them.

12.2 Admission under non-stationary task distributions

Evidence basis: extrapolated from deployment and focus/maintenance evidence. Admission and focus interact badly under non-stationary deployment: yesterday's verifier can become a poor predictor of today's utility, and yesterday's focused library can become harmful. Existing systems (SkillClaw, AutoSkill, MetaClaw) use eviction but do not adapt the verifier itself. The open problem is a verifier whose threshold and scoring function update from downstream utility.

12.3 Theoretically grounded maintenance schedules

Evidence basis: measured need, extrapolated schedule theory. Maintenance becomes important once libraries grow, but published schedules are engineering defaults. AutoRefine runs per-episode maintenance, MetaClaw/SKILL0/K2-Agent trigger distillation periodically or by prevalence, and AgentSkillOS audits on a human-scale cadence. The open problem is to derive when to invoke Prune, Merge, or Distill from monitored quantities such as admission rate, drift rate, retrieval entropy, and maintenance cost. Online learning with restarts is a plausible formal analogue.

Open problem	Evidence basis	Concrete first experiment	Likely obstacle
Compositional verifiers for code skills.	extrapolated	Replace one CODE-SHARP mutation with a PSN-style rollback-gated variant; measure velocity-vs-admission frontier.	Different graph semantics (invocation vs prerequisite).
Admission under non-stationary task 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.
Theoretically grounded maintenance schedules.	measured + extrapolated	Derive optimal maintenance frequency f^* from a cost model; compare to the periodic-distillation defaults used by two-timescale systems (MetaClaw, SKILL0).	Bounds likely depend on unobserved drift rate.
Parametric-collapse prevention under repeated distillation.	speculative	Run any two-timescale system for 10× its reported distillation windows; track proposal diversity.	Compute budget for long horizons.
Retrieval scaling beyond the flat-retrieval knee.	measured + extrapolated	Evaluate learned-routing, graph retrieval (Graph of Skills), skill-compilation, and hybrid-parametric baselines at 10,000–1,000,000 skills (e.g., against AgentSkillOS's 200K capability-tree curve).	Apples-to-apples comparison across storage structures is not yet standardized.
Cross-library skill portability.	measured + extrapolated	Extend CoEvoSkills / XSkill-style transfer into a controlled cross-product of author backbone, receiving backbone, harness, context budget, and tool API.	Tool / tokenizer / context mismatches; transfer can increase exploration while lowering average rollout quality.
Governance, provenance, and attribution.	measured + extrapolated	Scale ASG-SI-style audited provenance plus SkillSieve-style triage: log-based "logical undo" vs cryptographic provenance vs snapshot at 10^3 edits/day.	Provenance and scanner overhead under high operator velocity.
Benchmarks honest about dynamics.	measured	Combine SkillFlow-Bench's sequential patch protocol with Wild-Skills' retrieval realism; report global library trajectory, operator velocity, and usage-vs-utility.	Standardizing a shared global-library protocol across task families.

Table 10: **Eight open problems for the dynamic-skills research community, each paired with a concrete first experiment.** The first four rows are method-level problems tractable for a single team; the last four require infrastructure or community coordination. The "likely obstacle" column states the main technical blocker for planning a research program.

12.4 Preventing parametric-collapse under repeated distillation

Evidence basis: speculative. Two-timescale systems may risk a degenerative echo: each distillation window compresses skills into the model, and the model then proposes skills resembling what it just absorbed. No surveyed method (MetaClaw, SKILL0, K2-Agent) runs long enough to test this directly. The missing evidence is long-horizon behavior under repeated distillation, especially proposal diversity and dependence between new proposals and recently distilled skills.

12.5 Retrieval beyond the 128-skill knee

Evidence basis: measured at moderate scale, extrapolated to marketplace scale. Hierarchy, DAG, and ontology storage push the 64–128-skill retrieval knee rightward but do not prove general removal. The open problem is scaling composition to the 10,000–1,000,000-skill regime suggested by SkillRouter's 80K pool, AgentSkillOS's 200K-scale evaluation, SkillNet, and Skilldex. Graph of Skills (Liu et al., 2026a) improves dependency-aware retrieval up to 2,000 skills, but marketplace-scale evidence is missing. Three directions are ready for comparison: learned routing over skill IDs (SkillRouter), skill compilation via Merge/Distill (Single-Agent-Skills), and hybrid parametric–retrieval systems that retrieve only when routing confidence is low.

12.6 Cross-library skill portability and naming

Evidence basis: measured transfer cases, extrapolated compatibility theory. Skills increasingly move across agents and users (SkillClaw, AgentSkillOS, AutoSkill), but the literature treats portability as deployment rather than learning. The open problem is *capability compatibility*: when is a skill authored on agent A safe and useful on agent B? Tool API, tokenizer, context window, and harness assumptions can all break transfer. CoEvoSkills reports broad cross-model transfer, while XSkill reports mixed multimodal transfer; a compatibility theory should predict such outcomes before evaluation.

12.7 Governance, provenance, and attribution

Evidence basis: measured attacks and partial defenses, extrapolated integration. §11 shows fast growth in attack and measurement work but few integrated defended systems. The open problem is making ASG-SI-style audited graphs, evidence bundles, and verifier-gated promotion (Huang & Huang, 2025) compatible with 10^2–10^3 fast-loop edits per day and marketplace-grade triage. SkillSieve, Credential Leakage, and Malicious-or-Not (Hou & Yang, 2026; Chen et al., 2026e; Holzbauer et al., 2026) supply pieces of the admission scanner; missing are operator-level provenance and costed deployment architectures. Candidate mechanisms include logical undo over operator logs and cryptographic provenance linking each skill to the trajectory that justified admission.

12.8 Benchmarks that are honest about dynamics

Evidence basis: measured benchmark gaps. SkillFlow-Bench is the clearest temporal benchmark for discovery, patching, reuse, and repair; Wild-Skills is the clearest benchmark for realistic retrieval and curation utility. The open problem is to combine temporal patching with retrieval realism while reporting global library trajectory, operator velocity, and usage-vs-utility.

Together, the first four problems improve algorithms inside existing mechanism families, while 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.

13 Limitations and Update Policy

This survey makes analytical claims about a young area, so its limitations matter. The corpus is frozen at the April 24, 2026 cutoff in §2.3; later work may change the evidence for individual regularities, especially in registry-scale retrieval and safety. The taxonomy is therefore an updatable frame, not a final vocabulary.

The search protocol is broad but not exhaustive. The corpus was assembled through iterative search and snowballing over a fast-moving arXiv-heavy area, so false negatives are likely, especially for unpublished systems, product documentation, non-English papers, and papers that use tool, memory, workflow, or curriculum terminology without the word "skill". The cutoff should therefore be read as an audit boundary over the cited literature, not as a claim that every adjacent artifact-learning paper has been enumerated.

The evidence is heterogeneous. Method papers, benchmarks, infrastructure papers, and safety studies do not support the same kinds of claims: a verifier ablation is stronger evidence for an algorithmic regularity than a single benchmark score, while a registry-scale measurement is stronger evidence for a deployment surface than for a remedy. We use the lifecycle framework to align evidence roles rather than rank all papers on one axis.

The definition of "skill" remains unstable across executable programs, natural-language lessons, graphs, adapters, memories, and capability labels. Our six-sense terminology and 7-tuple formalism make the boundary explicit, but generic tool-use papers, memory-only agents, and fine-tuning pipelines are excluded unless they produce an externally invocable artifact or evaluate that artifact's lifecycle.

The lifecycle framing is also imperfect. It fits systems that externalize artifacts into stores, but purely parametric methods compress proposal, verification, and admission into a training loop; memory-only methods may retrieve episodes without ever forming an invocable interface; and multimodal or embodied systems may attach verification to a simulator, visual affordance, or physical rollout rather than to a textual artifact. We include such systems only when they expose enough artifact structure to compare lifecycle choices.

The operator algebra is intentionally coarse. It does not model every edit dependency, such as 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. The algebra is useful for comparing operator repertoires and storage costs, but it is not a full operational semantics for agent development environments.

Most conclusions are architectural rather than causal. The evidence supports claims such as "admission gates matter", "flat retrieval has a moderate-size failure mode", and "benchmarks under-report library trajectories", but not a universal prescription for the best verifier, storage topology, maintenance schedule, or distillation cadence. We also do not provide a quantitative cross-paper leaderboard: backbone, context budget, tool surface, evaluator, and task stream vary too much for pooled effect sizes to be meaningful without a shared harness.

14 Broader Impact Statement

Dynamic skill systems can make LLM agents more useful by preserving verified procedures, reducing repeated trial-and-error, improving transfer across related tasks, and exposing reusable artifacts for human inspection. These benefits matter most in operational settings where agents repeatedly use the same tools, interfaces, and domain workflows.

The same mechanism also creates new risks. A skill library is a high-trust execution substrate: admitted artifacts can carry prompt-injection text, unsafe code, bundled model backdoors, stale procedures, hidden credential leaks, or harmful but well-formed capabilities. Registry and marketplace settings add supply-chain, ownership, attribution, and skill-stealing concerns. This survey does not release new agent capabilities, but it organizes design patterns that could be used to build more autonomous skill-generation pipelines. The risk is highest if readers adopt growth and sharing mechanisms without the corresponding admission, containment, provenance, and rollback mechanisms analyzed in §11.

The mitigation message of the survey is therefore structural. Dynamic-skill research should treat safety and governance as lifecycle stages rather than after-the-fact filters: candidate skills should pass admission-time checks, carry manifests for permissions and dependencies, execute inside containment boundaries, retain operator-level provenance, and support demotion or rollback when downstream evidence changes. Benchmark papers should report unsafe invocation, skill usage versus skill utility, and maintenance-off behavior when those quantities are relevant to deployment.

15 Conclusion

When agents externalize reusable procedural knowledge, the skill library becomes part of the learning system. It has state, update rules, selection pressure, memory compression, maintenance costs, and safety constraints. Four claims about this object are now well supported; one remains unsupported.

The first supported claim is *the lifecycle view*. Dynamic skill systems are lifecycle-managed, verified, evolving artifact stores. The decomposition in §5 makes SkillWeaver, PSN, SkillRouter, MetaClaw, SkillFoundry, Skilldex, ClawSafety, and SkillFlow-Bench comparable without pretending that they solve the same problem.

The second supported claim is *the time index*. Viewing libraries as dynamic objects $\mathcal{L}_t$ makes maintenance-off ablations, library-size drops, verifier-quality sensitivity, usage-vs-utility gaps, and two-timescale decoupling expressible. The 7-tuple $\langle C, \pi, T, R, \varphi, \nu, \prec \rangle$ is the structural language for those analyses.

The third supported claim is *write-time discipline*. Admission, verifier quality under RL, maintenance, and write-time abstraction all concern what enters the library, under what gate, and with what abstraction. The evidence aligns with a broader lifelong-learning principle: what a system keeps can matter more than what it sees.

The fourth supported claim is *mechanism pluralism*. The operator algebra admits multiple coherent specializations: retrospective induction, execution-verified skill writing, skill-aware RL, cross-user aggregation, graph/hierarchy management, and two-timescale internalization. These are not separate taxonomies; they are different ways of coupling edit repertoire, admission, storage, and update clocks.

The unsupported claim is that scaling is solved. The moderate-library-size retrieval knee is recurrent; hierarchy, DAG, ontology, routing, and orchestration push it rightward but have not shown general removal. SkillFlow-Bench improves evaluation by tracking discovery, patching, repair, skill count, and usage, but its family-local protocol does not answer global heterogeneous-library scaling. Retrieval scaling, cross-library portability, provenance, and benchmark honesty remain open.

Reporting checklist. For future work to become comparable, papers should report five quantities:

1. which lifecycle stages are implemented and which are only assumed;
2. operator velocities and library trajectories over time, not only final task success;
3. repair-, maintenance-, or admission-off ablations when those stages are part of the method;
4. skill usage separately from skill utility;
5. the operators that an infrastructure or safety 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.

References

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

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

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

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

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

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

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

Yuxuan Cai, Yipeng Hao, Jie Zhou, Hang Yan, Zhikai Lei, Rui Zhen, Zhenhua Han, Yutao Yang, Junsong Li, Qianjun Pan, Tianyu Huai, Qin Chen, Xin Li, Kai Chen, Bo Zhang, Xipeng Qiu, and Liang He. Building Self-Evolving Agents via Experience-Driven Lifelong Learning: A Framework and Benchmark. *arXiv:2508.19005*, 2025.

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

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. SkillCraft: Can LLM Agents Learn to Use Tools Skillfully? *arXiv:2603.00718*, 2026b.

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. CUA-Skill: Develop Skills for Computer Using Agent. *arXiv:2601.21123*, 2026c.

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

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

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

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

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

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. A Comprehensive Survey of Self-Evolving AI Agents: A New Paradigm Bridging Foundation Models and Lifelong Agentic Systems. *arXiv:2508.07407*, 2025.

Loris Gaven, Thomas Carta, Clement Romac, Cedric Colas, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. MAGELLAN: Metacognitive Predictions of Learning Progress Guide Autotelic LLM Agents in Large Goal Spaces. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 18920–18953, 2025.

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

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

Yinghan Hou and Zongyou Yang. SkillSieve: A Hierarchical Triage Framework for Detecting Malicious AI Agent Skills. *arXiv:2604.06550*, 2026.

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. MedSkillAudit: A Domain-Specific Audit Framework for Medical Research Agent Skills. *arXiv:2604.20441*, 2026.

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

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

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

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

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

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

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

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

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

George Konidaris and Andrew G. Barto. Skill Discovery in Continuous Reinforcement Learning Domains Using Skill Chaining. In *Advances in Neural Information Processing Systems*, volume 22, pp. 1015–1023, 2009.

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

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. The World Won't Stay Still: Programmable Evolution for Agent Benchmarks. *arXiv:2603.05910*, 2026a.

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

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

Xiaoxiao Li. When Single-Agent with Skills Replace Multi-Agent Systems and When They Fail. *arXiv:2601.04748*, 2026.

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

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

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. SkillNet: Create, Evaluate, and Connect AI Skills. *arXiv:2603.04448*, 2026.

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

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

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

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

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

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

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. SkillClaw: Let Skills Evolve Collectively with Agentic Evolver. *arXiv:2604.08377*, 2026.

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

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

Zheng Nie, Ruolin Shen, Xinlei Yu, Bo Yin, Jiangning Zhang, and Xiaobin Hu. SkillGraph: Self-Evolving Multi-Agent Collaboration with Multimodal Graph Topology. *arXiv:2604.17503*, 2026.

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

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

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

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

Haochen Shi, Xingdi Yuan, and Bang Liu. Evolving Programmatic Skill Networks. *arXiv:2601.03509*, 2026.

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

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

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

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

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

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv:2305.16291*, 2023.

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

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

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

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

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

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

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

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

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

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

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

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

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

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. SkillRL: Evolving Agents via Recursive Skill-Augmented Reinforcement Learning. *arXiv:2602.08234*, 2026a.

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. MetaClaw: Just Talk – An Agent That Meta-Learns and Evolves in the Wild. *arXiv:2603.17187*, 2026b.

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

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

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. Learning on the Job: An Experience-Driven Self-Evolving Agent for Long-Horizon Tasks. *arXiv:2510.08002*, 2025a.

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

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. AutoSkill: Experience-Driven Lifelong Learning via Skill Self-Evolution. *arXiv:2603.01145*, 2026.

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

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. AgentEvolver: Towards Efficient Self-Evolving Agent System. *arXiv:2511.10395*, 2025.

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

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

Di Zhang. AgentDevel: Reframing Self-Evolving LLM Agents as Release Engineering. *arXiv:2601.04620*, 2026.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. CoEvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification. *arXiv:2604.01687*, 2026c.

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

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

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

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

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. SkillFlow: Benchmarking Lifelong Skill Discovery and Evolution for Autonomous Agents. *arXiv:2604.17308*, 2026h.

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. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. *arXiv:2504.07079*, 2025a.

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

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

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

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

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. Memento-Skills: Let Agents Design Agents. *arXiv:2603.18743*, 2026.

A Coding Protocol and Audit Materials

This appendix records the audit machinery behind the taxonomy. Method comparisons in the main text are based on mechanisms and evidence roles rather than on recency.

A.1 Screening and Coding Fields

Each note was coded for: problem framing, artifact type, update locus, trigger, operator repertoire, learning signal, storage topology, verification or admission mechanism, evaluation setting, headline results, ablations, limitations, closest peers, and survey takeaway. Papers were included in the primary method cluster if they changed a skill artifact or evaluated the lifecycle of such artifacts. Modern papers were treated as boundary/context within the broader audit set if they studied generic lifelong learning, generic tool use, memory-only agents, or adjacent self-evolution settings without an externally invocable skill artifact. Older options and hierarchical-RL papers were cited as background anchors, not coded as modern audit records.

The coding fields intentionally separate *what changes* from *how evidence supports the claim*. For example, a paper may be coded as an executable-library method because it edits code skills, while its evidence role may be benchmark, deployment measurement, or safety audit depending on what the paper actually measures. This separation is why the main text avoids cross-paper leaderboards unless the benchmark harness is shared.

A.2 Evidence Grades

The evidence grades used in Table 7 are qualitative audit labels, not statistical confidence intervals. Grade A indicates multiple controlled ablations, or one clean ablation plus independent corroboration. Grade B indicates one controlled study, a strong benchmark protocol, or a deployment-scale measurement. Grade C indicates convergent benchmark behavior without a clean causal ablation. Grade D indicates architectural corroboration only. Mixed labels such as A/B or B/C are used when the supporting papers differ in strength across artifact families.

A.3 Master Coding Sheet

Table 11 is the full representative coding sheet used to audit the lifecycle taxonomy. It is placed in the appendix because its role is reproducibility rather than narrative: the main text uses the family table and heatmap for synthesis, while this table gives reviewers a way to trace each representative system back to the seven coding fields.

Table 11: **Master coding table for representative dynamic-skill systems across the seven coding fields of Section 6, plus family cluster and one factual headline.** Cells use the legend below; the operator column uses single letters from the ten-element algebra of Equation 4. The table is a coding sheet that supports the lifecycle-family taxonomy rather than the primary conceptual taxonomy.
Legend. *Artifact:* Code = executable code, NL = natural-language heuristic, MD = SKILL.md, LoRA = parametric adapter, Mix = two or more. *Clock:* fast = in-context, slow = parametric, 2TS = two-timescale. *Trigger:* Task = task-end retrospective, Fail = failure-driven, Per = periodic maintenance, User = author/user edit, RL = inside RL loop. *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 = verifier/judge, XUser = cross-user aggregation, Teach = teacher distillation. *Storage:* flat, tree, DAG, graph, subsp, ontol.

Method	Cluster	Artif.	Clock	Trig.	Operators	Signal	Store	Headline
Foundational								
Voyager (Wang et al., 2023)	Found.	Code	fast	Task	A	Rew+Exec	flat	3.3× items; 15.3× tech-tree
LATM (Cai et al., 2023)	Found.	Code	fast	Task	A	Exec	flat	Tool-maker + tool-user beats few-shot

continued on next page

Table 11 (continued)

Method	Cluster	Artif.	Clock	Trig.	Operators	Signal	Store	Headline
Skill-aware RL								
SAGE (Wang et al., 2025)	RL	Code	2TS	Task	A,R	Rew	flat	Skill-integrated RL on AppWorld
SkillRL (Xia et al., 2026a)	RL	Code	slow	RL	A,D	Rew	tree	Hier. skill-conditioned RL
AgentEvolver (Zhai et al., 2025)	RL	Mix	2TS	RL	A,R,D	Rew+Crit	flat	Self-question / navigate / attribute
CODE-SHARP (Bornemann et al., 2026)	RL	Code	2TS	Fail	A,R,M,W	Rew+Judge	DAG	4 mutations + prereq DAG
Agentic-Prop. (Jiao et al., 2026)	RL	Mix	slow	RL	A,R,P,D,B,C	Teach+Judge	DAG	91.6% AIME-25 (30B, 11K traj.)
COS-PLAY (Wu et al., 2026a)	RL	Mix	2TS	RL	A,R,P,D	Rew	flat	Co-evolves decision + skill bank
Heuristic / lesson memory								
ERL (Allard et al., 2026)	Lesson	NL	fast	Task	A,R	Crit	flat	Task-end retrospective lessons
RetroAgent (Zhang et al., 2026e)	Lesson	NL	fast	Fail	A,R	Crit+Judge	flat	Dual intrinsic feedback
MUSE (Yang et al., 2025a)	Lesson	NL	fast	Per	A,R,D	Crit	tree	Long-horizon productivity memory
EvolveR (Wu et al., 2025)	Lesson	NL	fast	Task	A,R,W	Crit	flat	Strategic-principle evolution
Memento (Zhou et al., 2026)	Lesson	MD	fast	Task	A,R,K	Crit+Judge	tree	Case memory + SKILL.md rerank
XSkill (Jiang et al., 2026a)	Lesson	Mix	fast	Task	A,R,M,P,B,K	Crit+Judge	flat	Visual dual skill/exp. store
SAGER (Tao et al., 2026)	Lesson	NL	fast	User	A,R,W	XUser	flat	Per-user policy skills for rec.
Executable skill libraries								
SkillWeaver (Zheng et al., 2025a)	ExecLib	Code	fast	Task	A,R,P	Judge+Exec	flat	Propose-practice-synth.-hone
AgentFactory (Zhang et al., 2026g)	ExecLib	Code	fast	User	A,R,C	Judge	DAG	Subagents as evolvable skills
EvoSkill (Alzubi et al., 2026)	ExecLib	Code	fast	Fail	A,R,P	Judge+Exec	flat	Failure-driven + Pareto admission
CoEvoSkills (Zhang et al., 2026c)	ExecLib	MD	fast	Task	A,R,B	Judge	flat	Co-evolving verifier loop
PSN (Shi et al., 2026)	ExecLib	Code	fast	Fail	A,R,M,W,K,P	Crit+Judge	graph	5 refactors + symbolic credit
SkillCraft (Chen et al., 2026b)	ExecLib	Code	fast	Task	A,C,R	Exec+Judge	DAG	Verified compositional MCP
LIVE-SWE (Xia et al., 2025a)	ExecLib	Code	fast	Task	A,R	Exec	flat	Runtime scaffold synthesis
CASCADE (Huang et al., 2025)	Lesson	Mix	fast	Per	A,D,P	Teach	tree	Scientific skill consolidation
SkillX (Wang et al., 2026a)	ExecLib	Code	fast	Per	A,S,B,K	Judge	tree	Automatic hierarchical library
Trace2Skill (Ni et al., 2026)	ExecLib	MD	fast	Task	A,M,D	Judge	flat	Prevalence-weighted consolidation
SkillClaw (Ma et al., 2026)	ExecLib	MD	fast	User	A,R,M,K	XUser	flat	Cross-user skill evolution
AutoSkill (Yang et al., 2026)	ExecLib	MD	fast	User	A,R	XUser	flat	Training-free personalized bank
AutoRefine (Qiu et al., 2026)	ExecLib	MD	fast	Per	A,R,M	Crit+Judge	flat	Dual-form skills + maintenance
MemSkill (Zhang et al., 2026d)	ExecLib	MD	fast	Task	A,R,K	Crit	tree	Meta-memory skill banks
Wild-Skills (Liu et al., 2026d)	ExecLib	Code	fast	Per	A,P,K	Rew+Judge	flat	Utility under realistic retrieval
CUA-Skill (Chen et al., 2026c)	ExecLib	Code	fast	Task	A,B	Exec	flat	Parameterized GUI skills
ABSTRAL (Song et al., 2026)	Infra	MD	fast	Task	A,R,C	Crit	graph	Multi-agent design via SKILL.md

continued on next page

Table 11 (continued)

Method	Cluster	Artif.	Clock	Trig.	Operators	Signal	Store	Headline
SkillFoundry (Shen et al., 2026)	ExecLib	Mix	fast	Per	A,R,M,P,B	Exec+Judge	tree	Scientific contracts + provenance
SkillForge (Liu et al., 2026c)	ExecLib	MD	fast	Fail	A,R,W	Judge	flat	Failure diagnosis in cloud support
SkillMOO (Gong et al., 2026)	ExecLib	MD	fast	Fail	R,P,K	Judge+Exec	flat	Pareto pass/cost optimization
Bilevel-MCTS (Huang et al., 2026a)	ExecLib	MD	fast	Per	R,W	Judge	flat	MCTS over package structure
Metasurface (Huang et al., 2026b)	ExecLib	Mix	fast	Fail	A,R	Exec	flat	Physics-verified scientific skills
EvoAgent (Zhang et al., 2026a)	ExecLib	Mix	fast	User	A,R,K,C	XUser+Judge	tree	Multi-file skills + delegation
MACRO (Fan et al., 2026)	ExecLib	Code	2TS	Task	A,C,D	Rew+Exec	flat	Discovers composite medical tools
Parametric / training-time								
SELAUR (Zhang et al., 2026b)	Param	LoRA	slow	Fail	D	Rew(unc)	subsp	Uncertainty-reshaped fail-rewards
SKILL0 (Lu et al., 2026)	Param	LoRA	slow	Per	D,B	Teach	subsp	Helpfulness-decaying curricula
LSE (Chen et al., 2026d)	Param	Mix	slow	Task	D,R	Rew	subsp	Trained prompt-context evolution
SkillsCrafter (Wang et al., 2026d)	Param	LoRA	slow	Per	D,M,K	Rew	subsp	LoRA bank + semantic subspace
K2-Agent (Wu et al., 2026b)	Param	Mix	2TS	Task	A,R,D	Rew+Teach	tree	Decl.-proc. co-evolution
MetaClaw (Xia et al., 2026b)	Param	Mix	2TS	Per	A,R,D	Rew	flat	Fast-skill / slow-weight adapt.
Co-Evolving (Jung et al., 2025)	Param	Mix	slow	Fail	A,D,P	Rew+Crit	flat	Hard-negative failure training
Agent0 (Xia et al., 2025b)	Param	Mix	2TS	RL	A,D	Rew	flat	Zero-data curric.-executor co-ev
Tool-R0 (Acikgoz et al., 2026)	Param	Mix	slow	RL	A,D	Rew	flat	Dual self-play for tool-use
SCALAR (Zabounidis et al., 2026)	Param	Mix	slow	RL	D,B	Teach	flat	Co-adapt difficulty + env
ARISE (Li et al., 2026d)	Param	LoRA	slow	RL	D,K	Rew	subsp	Swarm PPO + PSO actions
EXIF (Yang et al., 2025b)	Param	LoRA	slow	RL	D,B	Rew+Teach	flat	Exploration-first skill data
Infrastructure & governance								
AgentSkillOS (Li et al., 2026b)	Infra	MD	fast	Per	A,C,S,P,K	Judge	tree	Capability tree + DAG orch.
SkillRouter (Zheng et al., 2026)	Infra	Code	fast	Per	K	Judge	flat	Full-text retrieval 80K skills
SkVM (Chen et al., 2026a)	Infra	Code	fast	User	A,R,C	Exec	DAG	Skills as compilable code
SkillNet (Liang et al., 2026)	Infra	MD	fast	Per	A,K,P	Judge	graph	Ontology + relation graph
SkillOrchestra (Wang et al., 2026b)	Infra	MD	fast	Per	K,C	Judge	ontol	Skill handbooks for routing
SkillFlow-2025 (Li et al., 2025)	Infra	MD	fast	Task	K	Judge	flat	Multi-stage community-skill retrieval
SkillFlow-Bench (Zhang et al., 2026h)	Eval	MD	fast	Task	A,R,P	Crit+Judge	flat	166 tasks / 20 task families
AutoAgent (Wang et al., 2026c)	Infra	Mix	2TS	Task	A,R,C	Crit+Judge	DAG	Evolves tools, peers, and self
AgentDevel (Zhang, 2026)	Infra	MD	fast	User	A,R,K,P	XUser	graph	Release engineering + lineage
GEA (Weng et al., 2026)	Infra	Mix	2TS	Per	A,R,M	Judge+Rew	ontol	Group-based framework evolution
Single-Ag.Sk. (Li, 2026)	Infra	Code	fast	Per	A,M,D,P	Judge	flat	Multi→single compilation

continued on next page

Table 11 (continued)

Method	Cluster	Artif.	Clock	Trig.	Operators	Signal	Store	Headline
EffiSkill (Wang et al., 2026f)	Infra	Code	fast	Per	A,B,K	Rew	flat	Mined operator skills
Graph of Skills (Liu et al., 2026a)	Infra	Code	fast	User	K,C	Rew	graph	Dependency-aware retrieval
GraSP (Xia et al., 2026c)	Infra	Code	fast	Task	C,R	Exec	DAG	Typed DAG composition + repair
Skilldex (Saha & Hemanth, 2026)	Infra	MD	fast	User	A,K,P	Judge	tree	Package manager + scoped registry
Corpus2Skill (Sun et al., 2026)	Infra	MD	fast	User	A,S,K	Judge	tree	Navigable enterprise QA skills
SkillRepoMining (Bi et al., 2026)	Infra	MD	fast	User	A,B	Judge	flat	GitHub repo mining to SKILL.md
AgentSkills-Data (Ling et al., 2026)	Eval	MD	fast	Per	K,P	Judge	—	40K+ public-skill ecosystem study
SkillLearnBench (Zhong et al., 2026)	Eval	MD	fast	Task	A,R	Judge	flat	Continual skill-generation benchmark
Safety & audit								
ASG-SI (Huang & Huang, 2025)	Safety	Mix	2TS	Task	A,R,P,D	Rew+Judge	graph	Audited skill graph + verif. rewards
ClawSafety (Wei et al., 2026)	Safety	MD	fast	User	P	Judge	—	Highest-ASR vector in personal agents
Secure-Skills (Li et al., 2026e)	Safety	MD	fast	User	P	Judge	—	Lifecycle threat taxonomy
Supply-Chain (Qu et al., 2026)	Safety	MD	fast	User	P	Judge	—	DDIPE poisoning attacks
SkillSieve (Hou & Yang, 2026)	Safety	MD	fast	User	P	Judge	—	Hierarchical malicious-skill triage
SkillAttack (Duan et al., 2026)	Safety	MD	fast	User	P	Judge	—	Automated attack-path red teaming
BadSkill (Tie et al., 2026)	Safety	Mix	fast	User	P	Exec	—	Model-in-skill backdoors
CredLeak (Chen et al., 2026e)	Safety	Mix	fast	User	P	Exec+Judge	—	Cross-modal secret leakage
HarmfulSkillBench (Jiang et al., 2026c)	Safety	MD	fast	User	P	Judge	—	Harmful-but-valid skills
SkillStealing (Wang et al., 2026e)	Safety	MD	fast	User	K	Judge	—	Black-box skill extraction
Malicious-or-Not (Holzbauer et al., 2026)	Safety	MD	fast	User	P	Judge	—	Repository-aware classification
MedSkillAudit (Hou et al., 2026)	Safety	MD	fast	User	P,R	Judge	—	Medical release-readiness audit