EVOHARNESSBENCH: CAN YOUR AGENTS KEEP PACE WITH AN EVOLVING HARNESS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern LLM-based agents rely on an evolving *harness* of tools, reusable skills, and specialist agents, yet existing evaluations largely assume this harness is fixed. We introduce **EVOHARNESSBENCH**, a benchmark for systematically evaluating agents under controlled harness evolution along three axes: *tools*, *skills*, and *agents*. Unlike existing continual-learning benchmarks for agents, which typically place non-stationarity in the task stream while keeping the harness fixed, EVOHARNESSBENCH places non-stationarity in the externally supplied harness itself. The benchmark contains 17 multi-stage harness streams constructed deterministically from verifier-based benchmarks, comprising **802 tasks, 520 tools, 42 skills, and 62 agents**. We evaluate two complementary settings that capture the central challenges of harness evolution: *deployment evaluation*, which measures whether agents retain previously accessible competence as the harness expands, and *self-evolving adaptation evaluation*, which tests whether accumulated experience remains useful as new capabilities are introduced. Our experiments reveal three persistent limitations in state-of-the-art agents. First, harness expansion alone can degrade performance on previously solved tasks, leading to *harness-induced forgetting*. Second, gains from self-evolving adaptation are inconsistent across stages, capability axes, and environments. Third, retention and adaptation can be in tension: preserving earlier competence does not necessarily improve adaptation to newly introduced capabilities, and vice versa. Together, these results establish harness evolution as a distinct and important challenge for building agents that can continuously exploit new capabilities while preserving previously effective behavior.[1]

1 INTRODUCTION

LLM-based agentic systems involve much more than a model and a prompt. They operate through a **harness**: the tools they can invoke, the skills or procedures they can reuse, and the other agents they can coordinate with (Gu, 2026; Zhou et al., 2026; Macedo, 2026; Ke et al., 2025a). This harness shapes both what an agent *observes* and what it can *do*. Importantly, the harness presented to an agent is not static. In deployed systems, it is inherently **non-stationary**, continually changing as new tools, skills, and specialist agents are introduced, updated, or retired. For example, the Agentforce ecosystem of Salesforce has expanded steadily[2] (Figure 2, green dashed line), while the history of OpenAI's public skills repository reflects a continuing stream of capability additions, updates, and removals[3] (Figure 2, green line). These trends suggest that harness evolution is not a peripheral edge case, but an increasingly central property of real-world agentic systems.

Harness evolution creates two distinct challenges. (1) **Retention**: as the externally supplied harness expands, an agent must continue to solve tasks that were previously supported. The capabilities required for those tasks may still be present, but they are now embedded within a larger pool of tools, skills, or agents, making previously successful behavior harder to recover and execute reliably. This can lead to **harness-induced forgetting**: the underlying model parameters remain unchanged, yet previously accessible competence degrades solely because the harness through which the agent acts has expanded (Figure 1,(B)). (2) **Adaptation under accumulated experience**: in systems with

[1] Datasets at https://anonymous-hf.com/a/xi3eupszyvsg/, https://anonymous-hf.com/a/6r7t8x11mnfv/, and https://anonymous-hf.com/a/s4p1uibxxc6d/.

[2] https://agentexchange.salesforce.com/explore/agentforce

[3] https://github.com/openai/skills

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

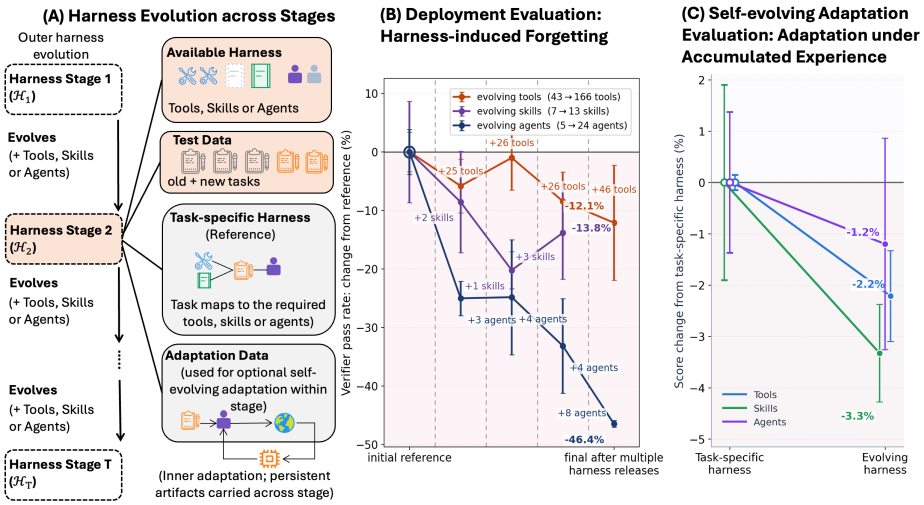


Figure 1: **(A)** EVOHARNESSBENCH evaluates agents under *outer harness evolution*, where the externally supplied harness evolves across stages ($\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_T$) along one capability axis: tools, skills, or specialist agents. At each stage, the *available harness* contains the full cumulative capability pool, while the *test data* include both newly introduced and previously encountered task cohorts. A *task-specific harness*, containing only the capabilities required for each task, serves as a reference condition. An *adaptation split* is used optionally for inner self-evolving adaptation, with persistent artifacts carried across stages. This yields two settings: *deployment evaluation*, without inner adaptation, and *self-evolving adaptation evaluation* with it. **(B)** Deployment evaluation reveals *harness-induced forgetting* from harness evolution alone. **(C)** Self-evolving adaptation evaluation shows that adapting under an evolving harness can be more challenging than under task-specific reference harnesses.

self-evolving adaptation[4], persistent artifacts such as memories, learned skills, prompts, or routing policies are accumulated under **earlier**, **narrower** harnesses. As new capabilities are introduced, these artifacts may become stale or even actively misdirect execution toward behaviors that were effective under previous harnesses but are no longer appropriate under the current one (Figure 1,(C)).

To expose and systematically evaluate these two challenges, we introduce **EVOHARNESSBENCH**, a benchmark specifically designed to study the impact of **harness evolution** on agent performance. To our knowledge, EVOHARNESSBENCH is the **first benchmark** to evaluate how growth in an externally supplied harness affects agent performance over time. Unlike prior benchmarks (Li et al., 2026; Zhong et al., 2026; Asawa et al., 2026), which typically keep the harness fixed, place non-stationarity only in the task stream, or study capability growth produced by the agent itself, EVOHARNESSBENCH makes the externally supplied harness the source of non-stationarity and measures how agent performance changes as that harness evolves. It constructs nested, cumulative harness streams along three axes: *tools* (expanding API catalogs), *skills* (growing libraries of reusable procedures), and *agents* (expanding pools of specialist agents). We introduce capabilities in stages, from frequently used core capabilities to the long tail, assigning each task to the earliest stage where all required capabilities are available, including at least one newly introduced capability.

In EVOHARNESSBENCH, we adopt two complementary evaluation modes, **deployment evaluation** and **self-evolving adaptation evaluation**, corresponding to the two challenges posed by harness evolution. Both are organized around **outer harness evolution**, where the externally supplied harness expands across stages as new tools, skills, or agents are introduced, with optional **inner self-evolving adaptation** within each stage (Figure 1A). In *deployment evaluation*, inner adaptation is disabled: each harness version is evaluated independently, with no experience carried across stages. This setting isolates the direct effect of harness evolution, testing whether frontier agents can leverage newly available capabilities while remaining robust to the growing pool of alternatives and potential distractions. In *self-evolving adaptation evaluation*, inner adaptation is enabled: at each stage, the system may learn through a self-evolving adaptation method and carry the resulting persistent artifacts, such as memories, learned skills, or prompts, across subsequent harness versions, while evaluation is performed on a held-out split. This setting measures whether experience accumulated under earlier harnesses continues to support effective adaptation as new capabilities are introduced.

[4]We focus on *harness-based* self-evolving adaptation: the underlying model weights remain fixed, while persistent components surrounding the model may be updated.

In total, EVOHARNESSBENCH contains *17 controlled harness streams*, each comprising *3 to 6 cumulative stages*. It includes *802 unique tasks*, instantiated as *1,510 axis-specific evaluation examples* spanning *520 executable tools*, *42 latent reference skills*, and *62 specialist agents*. We evaluate representative single-agent systems (SAS) and multi-agent systems (MAS) under both evaluation modes. Our experiments reveal three persistent gaps in state-of-the-art agents. (1) **Harness evolution is not free**: frontier agents can lose competence on previously solved tasks simply because the externally supplied harness has expanded, even when the capabilities required to solve those tasks remain available.(2) **Self-evolving adaptation remains inconsistent**: gains vary substantially across harness stages, capability axes, and environments, indicating that experience accumulated under earlier harnesses does not reliably transfer as the harness evolves. (3) **Retention and adaptation can compete**: methods that best preserve previously accessible competence may hinder adaptation to newly introduced capabilities, while methods that adapt effectively may sacrifice retention. Together, these findings show that current agents do not yet keep pace reliably with harness evolution. They motivate methods that explicitly couple *outer harness evolution* with *inner adaptation*, so that agents can preserve earlier competence while effectively using newly introduced capabilities.

2 RELATED WORK

Table 1: Comparison with related agent benchmarks.

Benchmark	Outer Evol.	Inner Adapt.	Primary Evaluation	Axes
Static Harness Benchmarks				
API-Bank / τ-bench	None	No	Task performance	Tools
SkillsBench / SkillCraft	None	No	Task performance	Skills
MultiAgentBench	None	No	Task performance	Agents
Benchmarks for Self-Evolving Agents				
Paper-/ RE-/ MLE-Bench	None	Yes	Task performance	–
SkillLearnBench / SkillFlow	None	Yes	Learned-skill quality	Skills
Continual Agent Benchmarks				
AgentCL / ContinualBench	Task stream	Yes	Continual adaptation	–
CMRBench	Model pool	Yes	Continual routing	Models
EVOHARNESSBENCH	**Harness**	**Optional**	**Deployment / Self-evolving adaptation**	**Tools, Skills, Agents**

Table 1 summarizes the existing agent benchmarks, none of which directly studies performance under an evolving harness (see Appendix B for details): (1) **Static-harness benchmarks** such as τ-bench, SkillsBench and MultiAgentBench evaluate agents with a fixed set of tools, skills, or agents (Li et al., 2023; Yao et al., 2024; Li et al., 2026; Zhu et al., 2025); they involve neither outer harness evolution nor inner adaptation. (2) **Self-evolving benchmarks** study whether agents improve through accumulated experience (Ouyang et al., 2026). These settings permit inner adaptation but do not include outer harness evolution, and therefore do not capture how accumulated experience behaves as the externally supplied harness changes. Skill-learning benchmarks such as SkillLearnBench (Zhong et al., 2026) and SkillFlow (Zhang et al., 2026) do involve an expanding skill surface, but that growth is agent-driven as part of the adaptation process itself; their primary focus is learned-skill quality rather than task performance under an externally evolving harness. (3) **Continual-learning benchmarks** include both outer evolution and inner adaptation, but place the source of non-stationarity in the task stream rather than in the harness (Shu et al., 2026; Asawa et al., 2026). The closest setting is CMRBench (Bell et al., 2026), which studies continual routing over an externally expanding model pool. In contrast, EVOHARNESSBENCH extends harness evolution to tools, skills, and specialist agents.

3 EVOHARNESSBENCH

3.1 OVERVIEW

We formalize harness evolution as follows. Let $\mathcal{H}_{1:T} \triangleq (\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_T)$ denote a sequence of harness states satisfying $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \cdots \subseteq \mathcal{H}_T$, where $\mathcal{H}_t$ is the set of capabilities exposed to the agent at stage t. Depending on the evolution axis, each capability may correspond to an executable tool, a procedural skill, or a specialist agent. The sequence $\mathcal{H}_{1:T}$ defines the *outer harness evolution*: capabilities may be added between stages, but the harness remains fixed during execution within a stage. Associated with each stage t is a task set $\mathcal{D}_t = \{(x_i, y_i, \mathcal{C}_i)\}_{i \in I_t}$, where x_i denotes the user task, y_i is its verifier-checkable target outcome, and $\mathcal{C}_i$ is a hidden ground-truth set of required capabilities used only for benchmark construction and analysis. Once introduced, the task instance (x_i, y_i) remains unchanged at subsequent stages; only the surrounding harness expands. Let t_i denote the stage at which task i is introduced. By construction, every task satisfies:

1. **Feasibility:** all capabilities needed to solve the task are available at introduction, i.e., $\mathcal{C}_i \subseteq \mathcal{H}_{t_i}$.

2. **New-capability pressure:** at least one required capability is newly introduced at that stage, i.e., $\mathcal{C}_i \cap (\mathcal{H}_{t_i} \setminus \mathcal{H}_{t_i-1}) \neq \emptyset$.

At each stage t, the corresponding task set $\mathcal{D}_t$ is partitioned into an adaptation split $\mathcal{D}_{t,\text{adapt}}$ and a held-out evaluation split $\mathcal{D}_{t,\text{eval}}$. Systems with self-evolving adaptation may use $\mathcal{D}_{t,\text{adapt}}$ for *inner adaptation*, whereas $\mathcal{D}_{t,\text{eval}}$ is reserved for verifier-based evaluation. During evaluation, the agent receives the full cumulative harness $\mathcal{H}_t$, rather than an oracle-pruned task-specific subset, so previously introduced capabilities remain available as either useful affordances or potential distractors.

We instantiate EVOHARNESSBENCH from tasks in EnterpriseOps-Gym (**EOG**) (Malay et al., 2026) and Agents' Last Exam (**ALE**) (Sun et al., 2026). EOG contributes stateful enterprise workflows with structured tool annotations, while ALE contributes diverse agentic tasks with rich software-tool environments. Together, they yield **17 harness streams** with 3–6 stages each, covering **802 unique tasks** instantiated as **1,510 axis-specific evaluation examples** across **520 tools**, **42 latent reference skills**, and **62 specialist agents**. Full benchmark statistics are provided in Appendix C.

3.2 CONSTRUCTION PIPELINE

Our construction follows two principles: (1) **Scalable**: given an agent benchmark, task-level capability annotations are either inherited when available or derived deterministically from its procedural and tool structure. Our construction pipeline can therefore be generalized to others existing agent benchmarks to turn them into evolving tool, skill, or agent streams without AI-generated tasks or manual annotations or verification; (2) **Realistic**: the construction preserves the original tasks, environments, and execution interfaces, changing only the surrounding harness, while capabilities are released from frequently used core capabilities toward the long tail, a growth pattern that qualitatively aligns with real-world agent ecosystems (Figure 2). Following these principles, the construction pipeline is applied independently to each evolution axis $a \in \{\text{tool}, \text{skill}, \text{agent}\}$; for clarity, we omit the axis superscript below. Figure E.1 provides an overview of this construction process.

Step 1: Capability annotation. Each task i is associated with an axis-specific capability set $\mathcal{C}_i$. For the tool axis, these annotations are inherited directly from the oracle tool labels provided by the source benchmarks. For the skill and agent axes, where such annotations are not available, we construct them deterministically following the procedures we describe in Section 3.3.

Step 2: Frequency-ranked capability release. We define the capability[5] universe as $\mathcal{U} = \bigcup_i \mathcal{C}_i$ and measure the frequency of each capability $c \in \mathcal{U}$ as $f(c) = |i : c \in \mathcal{C}_i|$. We rank capabilities by their frequency across tasks in a source dataset and partitioned into T release buckets, which induces a release stage $r(c)$ for each capability. We introduce frequently used capabilities in the earlier stages, forming the *core*, while less frequent capabilities that form the *long tail* are released in later stages. Our ordering design thus reflects a common deployment pattern in which broadly useful capabilities are exposed before more specialized ones.

Step 3: Cumulative harness construction. The harness available at stage t is defined as $\mathcal{H}_t := \{c \in \mathcal{U} : r(c) \leq t\}$. By construction, the harness grows monotonically, i.e., $\mathcal{H}_t \subseteq \mathcal{H}_{t+1}$, and capabilities are never removed or modified after they are introduced.

Step 4: Task assignment. We assign each task to the earliest stage at which all of its required capabilities are available: $t_i := \min\{t : \mathcal{C}_i \subseteq \mathcal{H}_t\} = \max_{c \in \mathcal{C}_i} r(c)$. This directly guarantees the two properties stated in Section 3.1: *feasibility* and *new-capability pressure*.

3.3 AXIS-SPECIFIC CONSTRUCTION

Tools. The tool axis is the most direct: each task's capability annotation $\mathcal{C}_i^{\text{tool}}$ is given by the oracle tool set from the source benchmark. Applying the construction pipeline yields a nested sequence of tool catalogs, $\mathcal{H}_1^{\text{tool}} \subseteq \cdots \subseteq \mathcal{H}_T^{\text{tool}}$. At stage t, the agent receives the full cumulative tool catalog rather than an oracle-pruned task-specific subset, so later stages introduce both newly useful tools and a growing set of potential distractors.

[5]We use 'capability' to refer to an individual component of the evolving outer harness, such as a tool, skill, or specialist agent.

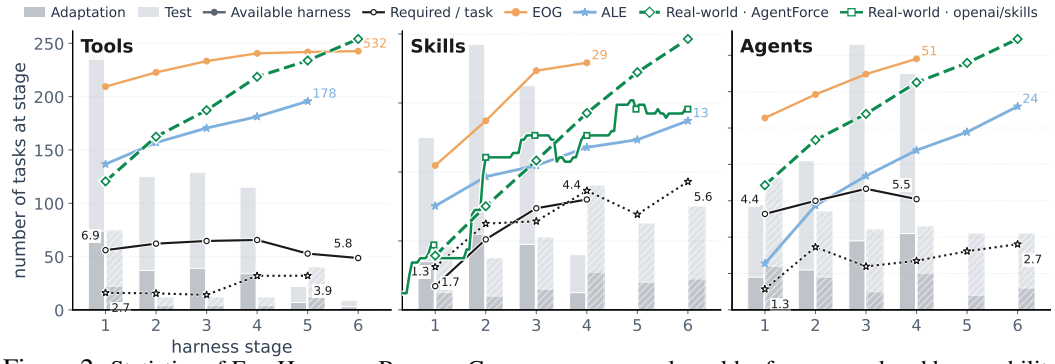


Figure 2: Statistics of EVOHARNESSBENCH. Green curves are real-world references ordered by capability release date. EVOHARNESSBENCH follows the same qualitative pattern: the available harness grows substantially over time while individual tasks require only a small subset of it.

Skills. Unlike tools, procedural skills are not typically annotated in the source benchmarks. We therefore construct $\mathcal{C}_i^{\text{skill}}$ deterministically in two steps:

1. *Mining reference skills.* Source system prompts typically contain both general instructions (e.g., role definitions and output-format constraints) and **reusable step-by-step procedures** for specific operations. We separate these components: general instructions remain in the prompt visible to evaluated systems, while procedural content is extracted into a set of hidden reference skills. This extraction is rule-based rather than LLM-generated.
2. *Associating skills with tasks.* Task verifiers operate on final states (e.g., database entries or file contents). We extract keywords from these verifier-checked states and match them against the hidden reference skills. A skill is associated with a task when entities or identifiers appearing in the verifier-checked state also appear in the skill's procedural content. The resulting matched set defines $\mathcal{C}_i^{\text{skill}}$, which we refer to as the task's *essential-skill annotations*. This keyword-based association is a heuristic proxy rather than an exact correspondence: it identifies skills related to the verifier-checked outcome, but does not guarantee that the matched skills are necessary, sufficient, or unique for solving the task.

We apply the same frequency-ranked release construction (Section 3.2) which yields $\mathcal{H}_1^{\text{skill}} \subseteq \cdots \subseteq \mathcal{H}_T^{\text{skill}}$. At each stage, the mined skills form the retrievable skill pool, but are not directly injected into the agent prompt. Instead, the original procedural content is removed from the prompt, requiring the system to identify and retrieve relevant skills from the expanding pool rather than rely on prompt-embedded procedures. We detail the skill-mining and task-matching procedure in Appendix E.

Agents. The agent axis transforms the tool structure into an expanding pool of specialist agents. We define an ownership map $\eta : \mathcal{U}^{\text{tool}} \to \mathcal{E}$ that assigns each tool to the entity it operates on (e.g., the database table storing user or case records, or stateful object such as a file or email record, as specified by its interface). Tools with the same owner are grouped into a bundle, and each bundle defines an entity-scoped specialist agent g_e, , such as a database specialist g_{database}, file-system specialist $g_{\text{filesystem}}$, or email specialist g_{email}, equipped with the corresponding tools and relevant procedural context identified during skill mining. Additional details are provided in Appendix F.

A task's agent annotation is defined as $\mathcal{C}_i^{\text{agent}} := \{g_{\eta(u)} \mid u \in \mathcal{C}_i^{\text{tool}}\}$. When the tools required by a task span multiple entities (e.g., finding a user in the database and then drafting and sending that user an email), the task therefore structurally requires delegation across multiple specialist agents. Applying the frequency-ranked release construction (Section 3.2) to these task-level annotations yields $\mathcal{H}_1^{\text{agent}} \subseteq \cdots \subseteq \mathcal{H}_T^{\text{agent}}$. During evaluation, a tool-less lead agent receives the cumulative pool $\mathcal{H}_t^{\text{agent}}$ and must select, delegate to, and coordinate specialists for each task.

3.4 EVALUATION PROTOCOL

Given a harness stream $\mathcal{H}_{1:T}$ and its associated task sets, we evaluate systems under two complementary modes that isolate distinct aspects of performance under harness evolution. We report task-level performance and operating cost, together with trajectory-level transfer metrics that distinguish adaptation to newly introduced tasks from retention of previously introduced tasks. Specifically, **Pass** is the fraction of tasks whose verifier reports successful completion, while **Score** is the average verifier score returned by the source benchmark, capturing partial credit when available. We use relative Forward Transfer (**FWT**) to measure adaptation to the current-stage cohort and relative Backward

Table 2: Main results on **evolving tools**. ReAct is used on EOG and Codex on ALE; 🖧 denotes multi-agent systems. All systems use GPT-5 to enable controlled comparison across harness and adaptation methods.

	EvoHarnessBench-EOG (454 tasks)				EvoHarnessBench-ALE (63 tasks)				Overall
	Pass (%)	Score	⏱ (h)	Tok. (M)	Pass (%)	Score	⏱ (h)	Tok. (M)	Pass Rate
(Task-specific Reference) Frontier Deployment Systems									
ReAct / Codex	26.0$^{\pm 0.6}$	57.0$^{\pm 1.6}$	48.3	27.2	11.6$^{\pm 1.5}$	34.6$^{\pm 3.4}$	9.7	82.2	24.2
Frontier Deployment Systems									
ReAct / Codex	30.2$^{\pm 0.8}$	60.0$^{\pm 0.9}$	28.6	75.8	12.2$^{\pm 0.7}$	30.0$^{\pm 1.7}$	9.2	100.8	28.1
🖧 AutoGen	30.8$^{\pm 0.1}$	62.6$^{\pm 0.2}$	24.4	63.7	11.3$^{\pm 2.3}$	32.2$^{\pm 2.4}$	6.5	19.6	28.6
Memory-based Self-evolving Adaptation									
Raw Memory	33.0$^{\pm 0.9}$	66.2$^{\pm 0.6}$	25.7	124.5	7.9$^{\pm 2.2}$	21.4$^{\pm 3.6}$	8.4	184.3	29.9
Reasoning Bank	36.9$^{\pm 1.3}$	68.7$^{\pm 0.4}$	21.1	150.8	11.1$^{\pm 1.3}$	28.4$^{\pm 1.5}$	10.0	106.8	33.8
MemToolAgent	38.6$^{\pm 1.0}$	68.9$^{\pm 0.5}$	23.9	148.0	10.1$^{\pm 2.0}$	25.8$^{\pm 1.2}$	13.1	83.8	**35.1**
🖧 G-Memory	32.2$^{\pm 1.2}$	64.5$^{\pm 0.6}$	30.5	66.0	13.9$^{\pm 0.1}$	32.5$^{\pm 1.6}$	7.4	20.9	30.1
🖧 LegoMem	22.2$^{\pm 0.7}$	57.3$^{\pm 0.4}$	52.8	141.7	8.3$^{\pm 1.7}$	23.7$^{\pm 2.7}$	13.8	39.5	20.6
🖧 DeLM	20.3$^{\pm 1.8}$	51.4$^{\pm 1.5}$	97.7	179.1	6.2$^{\pm 1.8}$	23.7$^{\pm 1.5}$	16.7	30.9	18.7
Prompt-based Self-evolving Adaptation									
GEPA	31.9$^{\pm 0.8}$	65.9$^{\pm 0.3}$	31.9	100.2	11.1$^{\pm 0.0}$	30.9$^{\pm 1.4}$	9.6	86.9	29.3
Code-based Self-evolving Adaptation									
Meta-Harness	35.2$^{\pm 0.7}$	65.8$^{\pm 1.1}$	27.3	99.0	11.1$^{\pm 1.3}$	30.4$^{\pm 1.3}$	11.2	107.8	32.3

Transfer (**BWT**) to measure changes in performance on earlier cohorts as the harness expands (Wang et al., 2024). In the following, we define the two evaluation modes. Full metric definitions and a visual summary of the evaluation modes are provided in Appendix D.

Deployment evaluation: the system carries no persistent state across stages. At each stage t, we instantiate a fresh system with harness $\mathcal{H}_t$ and evaluate it on the cumulative evaluation set $\mathcal{D}_{\leq t,\text{eval}} := \bigcup_{\tau=1}^{t} \mathcal{D}_{\tau,\text{eval}}$. For task x_i, we denote the system output by $\mathcal{A}(x_i; \mathcal{H}_t)$. This setting isolates the *direct effect of harness expansion*: whether an agent can exploit newly available capabilities while remaining robust to the growing pool of alternatives, without any benefit or interference from accumulated experience. Any degradation observed in this mode is therefore **harness-induced**: the model parameters and prompting remain fixed, and only the exposed capability set changes.

Self-evolving adaptation evaluation: the system maintains a persistent adaptive state z_t that evolves across stages. While $\mathcal{H}_t$ denotes the externally supplied harness, z_t captures the agent's accumulated knowledge for operating within that harness, such as episodic memories of prior tool-use patterns, self-generated procedural skills, or learned routing policies for delegating to specialist agents. We denote the system output at stage t by $\mathcal{A}(x_i; \mathcal{H}_t, z_t)$. Before evaluation at each stage, the system may use the cumulative adaptation set $\mathcal{D}_{\leq t,\text{adapt}} := \bigcup_{\tau=1}^{t} \mathcal{D}_{\tau,\text{adapt}}$ to update z_t. We describe the specific adaptation strategies considered in Section 4. This setting measures whether accumulated experience supports adaptation to newly introduced capabilities while preserving previously accessible competence, or instead becomes stale, misleading, or damages performance as the harness evolves.

4 Main Results and Analysis

Evaluation setup. We organize the results by evolution axis (*tools*, *skills*, and *agents*) and progress from deployment and self-evolving adaptation to transfer and mechanism analyses. Under **deployment evaluation**, we consider representative SoTA **single-agent systems (SAS)** including ReAct (GPT-5), Codex, and Claude Code (Sonnet-4.6)[6], as well as **multi-agent systems (MAS)** including AutoGen (Wu et al., 2024) and DeLM (Mao & Mirhoseini, 2026). For **self-evolving adaptation**, we instantiate the persistent state z_t (see Section 3.4) using three families of methods: **memory-based** approaches (Raw Memory[7], ReasoningBank (Ouyang et al., 2026), MemToolAgent (Er et al., 2026), G-Memory (Zhang et al., 2025a), and LEGOMem (Han et al., 2025)), which store and retrieve episodic experience; **prompt-based** adaptation (GEPA (Agrawal et al., 2026)), which updates persistent textual instructions from task feedback; and **code-based** adaptation (Meta-Harness (Lee et al., 2026)), which optimizes the executable harness itself. Within each comparison, we hold the underlying model and execution setting fixed so that observed differences can be attributed to harness

[6]Claude Code uses a different underlying model and harness from the controlled GPT-5-based systems. We therefore omit it from the main comparison tables and report its full results in Appendix H.

[7]Raw Memory is a minimal baseline that directly retains past interaction trajectories without consolidation or learned transformation. See details in Section G.1

Table 3: Main results on **evolving skills**. We include only SAS, as existing work benchmarking MAS in evolving skills settings remains limited. All systems use the GPT-5 model.

	EvoHarnessBench-EOG (148 tasks)				EvoHarnessBench-ALE (68 tasks)				Overall
	Pass (%)	Score	(h)	Tok. (M)	Pass (%)	Score	(h)	Tok. (M)	Pass Rate
(Task-specific Reference) Frontier Deployment System									
Codex	$18.9^{\pm 1.1}$	$57.8^{\pm 0.2}$	3.8	29.9	$7.4^{\pm 2.1}$	$27.9^{\pm 1.3}$	11.4	80.1	15.3
Frontier Deployment Systems									
Codex	$18.9^{\pm 0.6}$	$59.1^{\pm 1.1}$	3.7	31.0	$8.3^{\pm 0.7}$	$25.5^{\pm 0.5}$	10.8	82.7	15.6
Memory-based Self-evolving Adaptation									
Codex Memory	$17.8^{\pm 2.2}$	$58.0^{\pm 1.0}$	4.6	41.5	$9.3^{\pm 1.8}$	$25.6^{\pm 3.5}$	11.6	87.8	15.1
Raw Memory	$21.6^{\pm 1.0}$	$61.5^{\pm 0.3}$	2.8	93.0	$7.4^{\pm 0.0}$	$20.2^{\pm 1.6}$	12.2	156.4	17.1
Reasoning Bank	$20.9^{\pm 2.0}$	$60.2^{\pm 1.2}$	3.2	76.1	$8.3^{\pm 1.8}$	$23.4^{\pm 4.1}$	12.2	82.3	16.9
MemToolAgent	$19.6^{\pm 0.6}$	$59.2^{\pm 0.5}$	2.9	97.0	$6.9^{\pm 2.5}$	$22.6^{\pm 3.7}$	11.6	79.6	15.6
Prompt-based Self-evolving Adaptation									
GEPA	$24.1^{\pm 1.4}$	$61.0^{\pm 0.8}$	4.1	44.2	$7.4^{\pm 2.1}$	$27.8^{\pm 0.6}$	11.7	97.8	**18.8**
Code-based Self-evolving Adaptation									
Meta-Harness	$19.4^{\pm 1.9}$	$59.5^{\pm 1.0}$	4.0	36.5	$7.8^{\pm 1.8}$	$26.1^{\pm 2.4}$	11.3	89.9	15.7

evolution and the adaptation strategy. All evaluations also include a **task-specific reference** in which each task is exposed only to its annotated required capabilities (Section 3.2), rather than the full cumulative harness. This provides a controlled comparison between task-specific capability exposure and the broader cumulative harness.[8]

Evolving Tools. **Obs. ❶ (Deployment): The cumulative tool set outperforms the task-specific reference, but at substantially higher cost.** As shown in Table 2, compared with the task-specific reference, exposing frontier agents to the full cumulative tool catalog surprisingly improves pass rate on both EOG and ALE, but substantially increases token usage. Thus, broader tool access can improve deployment accuracy while making execution considerably more expensive. This suggests that the larger catalog is not merely a source of distraction: agents can benefit from additional affordances, but only through substantially more search and execution.

Obs. ❷ (Self-evolving adaptation): Additional gains are possible but inconsistent. Table 2 further shows that, on EOG, several methods substantially outperform the cumulative deployment baseline: MemToolAgent reaches 38.6% pass rate, followed by ReasoningBank (36.9%) and Meta-Harness (35.2%), compared with 30.2% without adaptation. On ALE, however, most methods remain close to or below the deployment baseline. This contrast shows that accumulating persistent experience does not reliably translate into better performance under harness evolution; its benefit varies substantially across environments.

Evolving Skills. **Obs. ❶ (Deployment): The cumulative skill set performs similarly to the task-specific reference.** In contrast to evolving tool libraries, moving from task-specific to cumulative skill exposure has almost no effect on deployment performance (Table 3). On EOG, Codex remains at a 18.9% pass rate, with only a modest increase in token usage. This suggests that expanding the skill library introduces little direct interference, likely because skills must be explicitly retrieved and invoked, rather than being continuously exposed to the model as executable actions.

Obs. ❷ (Self-evolving adaptation): Gains are modest and method-dependent. Given the limited effect of skill expansion itself, the more important question is whether adaptation can learn to use the available skills more effectively. As shown in Table 3, GEPA delivers the clearest improvement on EOG, increasing pass rate from 18.9% to 24.1%, while the other methods yield smaller or less consistent gains across environments. In contrast to tools, where several memory-based methods benefit from accumulated experience, effective skill adaptation appears to depend more strongly on learning *when* and *how* to retrieve and apply the provided skills.

Evolving Agents. **Obs. ❶ (Deployment): The cumulative agent pool can outperform or underperform the task-specific reference depending on the environment.** Unlike tools and skills, expanding the agent pool does not induce a consistent deployment trend (Table 4). Performance is already low under the task-specific reference pool, underscoring the intrinsic difficulty of multi-agent

[8] We refer to the resulting evolving-harness instances as EvoHarnessBench-EOG and EvoHarness-Bench-ALE; and shorten these to EOG and ALE when unambiguous. Task sets differ across axes because each axis requires different capability annotations. In EOG, the skill and agent tracks use CSM, HR, and ITSM, while the tool track covers all domains. Thus, absolute scores are not directly comparable across axes; our cross-axis analysis compares within-track effects and transfer patterns.

Table 4: Main results on **evolving agents**. We include only MAS, as this axis inherently requires coordination over a pool of specialist agents. "–" denotes experiments that could not be completed within a comparable computational budget. All systems use the GPT-5 model.

	EvoHarnessBench-EOG (148 tasks)				EvoHarnessBench-ALE (63 tasks)				Overall
	Pass (%)	Score	(h)	Tok. (M)	Pass (%)	Score	(h)	Tok. (M)	Pass Rate
(Task-specific Reference) Frontier Deployment System									
Codex	$6.5^{\pm 1.8}$	$36.3^{\pm 1.6}$	24.8	570	$5.3^{\pm 3.0}$	$19.2^{\pm 2.9}$	27.4	20.0	6.2
Frontier Deployment System									
Codex	$8.8^{\pm 4.5}$	$42.9^{\pm 3.0}$	25.3	587	$4.2^{\pm 2.0}$	$16.4^{\pm 1.3}$	37.8	30.3	7.4
Memory-based Self-evolving Adaptation									
Codex Memory	$10.1^{\pm 0.0}$	$42.3^{\pm 0.6}$	20.2	197	$3.4^{\pm 0.7}$	$15.4^{\pm 2.2}$	33.5	37.4	6.8
G-Memory	$18.2^{\pm 1.1}$	$55.1^{\pm 0.7}$	29.4	30.4	–	–	–	–	–
LEGOMemory	$13.3^{\pm 0.6}$	$44.7^{\pm 1.4}$	16.1	21.8	$5.5^{\pm 0.9}$	$18.9^{\pm 2.2}$	13.6	34.0	11.0
DeLM	$14.2^{\pm 1.5}$	$53.7^{\pm 1.1}$	30.2	32.1	–	–	–	–	–
Prompt-based Self-evolving Adaptation									
GEPA	$14.9^{\pm 0.6}$	$54.1^{\pm 0.9}$	16.5	172	$6.3^{\pm 1.3}$	$23.6^{\pm 0.9}$	28.2	25.9	12.3
Code-based Self-evolving Adaptation									
Meta-Harness	$18.5^{\pm 2.5}$	$57.6^{\pm 1.2}$	17.9	228	$3.2^{\pm 0.0}$	$18.3^{\pm 3.0}$	40.9	29.4	13.9

coordination. Moving to the cumulative pool then improves Codex on EOG ($6.5\% \rightarrow 8.8\%$) but degrades it on ALE ($5.3\% \rightarrow 4.2\%$). Thus, broader access to specialist agents is neither uniformly beneficial nor uniformly benign; its impact depends strongly on the environment-specific demands of agent selection, delegation, and orchestration.

Obs. ❷ (Self-evolving adaptation): Agent evolution yields the largest gains, but they are highly environment-dependent. On EOG, self-evolving adaptation produces the strongest relative improvements across the three harness axes: Meta-Harness increases pass rate from the 8.8% deployment baseline to 18.5%, with several other methods also delivering substantial gains (Table 4). This suggests that persistent adaptation has particularly high leverage in the more complex agent-coordination setting, where experience can improve selection, delegation, and orchestration. The pattern does not carry over to ALE, however, where most methods remain at or below the 4.2% deployment baseline and only GEPA shows a clear improvement.

4.1 Can Agents Keep Pace with Harness Evolution? A Cross-Axis Analysis

We show that moving from the task-specific reference to a broader cumulative capability set can help, hurt, or have little effect depending on the harness axis and environment. Self-evolving adaptation can further improve performance, but its gains are similarly inconsistent. These comparisons alone do not establish whether agents keep pace as the harness evolves: doing so requires adapting to newly introduced capabilities while retaining previously accessible competence. We therefore examine transfer across the harness trajectory, how the capability set used during adaptation shapes learning, and the remaining bottlenecks for tools, skills, and agents.

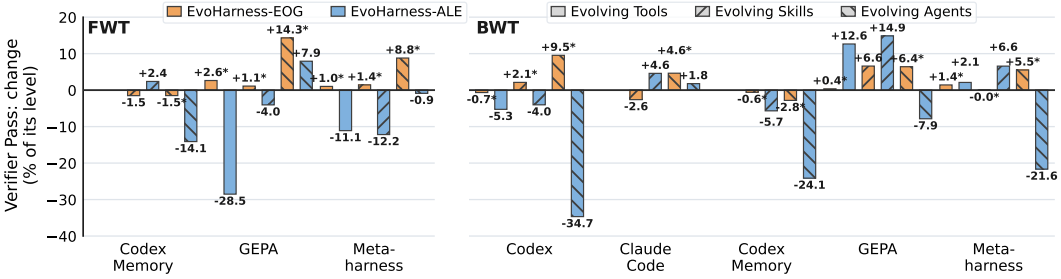


Figure 3: Average forward and backward transfer. $*$ indicates mixed transfer directions among domains.

Transfer: adaptation and retention do not necessarily move together. As shown in Figure 3, FWT and BWT expose behavior that aggregate performance obscures. Under evolving tools, deployment already exhibits harness-induced forgetting, while self-evolving adaptation can improve retention at the cost of adaptation to new capabilities: on ALE, GEPA and Meta-Harness obtain positive BWT but FWT drops to -28.5% and -11.1%, respectively. Evolving skills exhibit milder forgetting, yet the same tension persists: GEPA reaches $+14.9\%$ BWT on ALE while its FWT remains negative (-4.0%). Agent evolution is the most environment-dependent: adaptation can improve both FWT and BWT on EOG, whereas deployment BWT reaches -34.7% on ALE, the strongest harness-induced

forgetting across the three axes. Thus, keeping pace requires jointly optimizing adaptation to newly introduced capabilities and retention of previously accessible competence.

Capability-set variations: disentangling harness breadth from evolution. We ask whether adaptation is challenging because the harness is *broad*, because it *evolves over time*, or both. To disentangle these effects, we compare three adaptation settings while keeping evaluation under the same cumulative harness (Figure 4): (1) the standard *evolving cumulative harness*, where capabilities are introduced progressively; (2) a *task-specific harness*, which exposes only the capabilities required by each task; and (3) a *static full harness*, which provides the final cumulative capability set from the beginning.

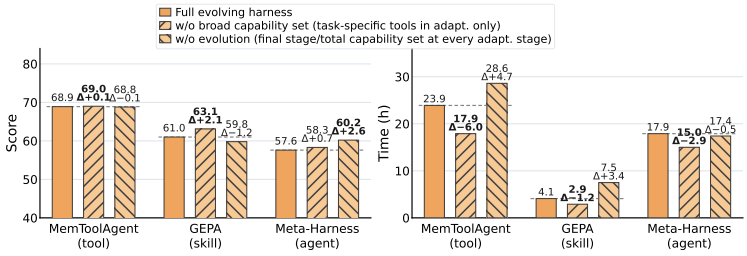


Figure 4: Effect of capability-set variations during adaptation.

For **tools**, restricting adaptation to task-specific capabilities achieves similar performance while substantially reducing adaptation cost, suggesting that the breadth of the harness itself contributes to adaptation difficulty. **Skills** show a similar reduction in adaptation time, although task-specific exposure has a mixed effect on performance. For **agents**, both alternative settings outperform the evolving harness, with the static full harness performing best, suggesting that changes in the agent pool itself are an important source of difficulty. Overall, these results indicate that both harness breadth and harness evolution can hinder adaptation, but their effects differ across capability types.

Persistent gaps are axis-specific. For tools, the key challenge is using past experience to constrain search over an expanding action space. Structured memories such as MemToolAgent and ReasoningBank outperform raw replay, while richer tool information can introduce interference, highlighting the importance of extracting actionable experience (Appendix I). For skills, availability does not imply utilization (Figure 5): GPT-5-based systems invoke few offered skills by default, with invocation varying substantially across models and adaptation policies. Grounded execution experience and iterative feedback are important, while poorly learned skills can underperform having no learned skills (Appendix J). For agents, self-evolution improves delegation completeness more than selection precision (Figure 6), with remaining errors increasingly reflecting coordination and execution after delegation (Appendix K). These results show that harness evolution poses axis-specific challenges: adaptation can yield substantial gains, but neither retention nor adaptation to new capabilities is reliable across axes and environments.

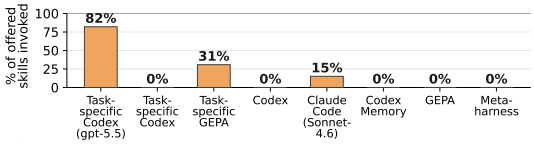


Figure 5: Share of offered skills invoked at the final stage across models and adaptation settings. GPT-5 and EOG are used unless otherwise specified.

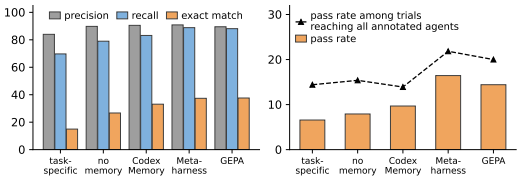


Figure 6: Delegation quality and task success on EOG.

5 Conclusion

We introduced EvoHarnessBench, a benchmark for evaluating agents under *harness evolution*: controlled growth of the externally supplied tools, skills, and agents available to a fixed underlying model. By separating outer harness evolution from optional inner self-evolving adaptation, EvoHarnessBench evaluates both deployment under a changing harness and whether persistent adaptation helps systems keep pace. By evaluating systems across harness stages rather than only at the final stage, EvoHarnessBench exposes adaptation and retention failures that aggregate evaluation can obscure. A promising future direction is to formulate harness evolution and agent adaptation as a bi-level optimization problem, with the outer level modeling evolving capabilities and the inner level adapting the agent to the evolving harness, enabling their co-evolution.

AI USE STATEMENT

In this work, we used generative AI tools for grammar checking, language editing and writing refinement in the manuscript. We did not use generative AI tools for dataset generation or other activities requiring disclosure under the ICLR 2027 AI policy. We have reviewed all AI-assisted edits and take full responsibility for the final content of this work, including its text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/0e9e708b6f48e14fd0ac29e167413f76-Paper-Conference.pdf.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier ai systems in real-world stateful environments, 2026. URL https://arxiv.org/abs/2606.05661.

Jack Bell, Giacomo Carfi', Gerlando Gramaglia, and Vincenzo Lomonaco. Continual model routing in evolving model hubs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=IT7lA8t0S6.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. URL https://arxiv.org/abs/2410.07095.

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?, 2026. URL https://arxiv.org/abs/2603.00718.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery, 2025. URL https://arxiv.org/abs/2410.05080.

Suleyman Armagan Er, Danilo Ribeiro, Yogesh Virkar, Surafel Lakew, Adi Kalyanpur, James Gung, Thomas Delteil, and Arshit Gupta. Memtoolagent: Leveraging memory for tool using agents based on environment and user feedback. *arXiv preprint arXiv:2606.07909*, 2026.

Matthias Galster, Seyedmoein Mohsenimofidi, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, and Sebastian Baltes. Harness engineering for agentic ai coding tools: An exploratory study, 2026. URL https://arxiv.org/abs/2602.14690.

Shangding Gu. From model scaling to system scaling: Scaling the harness in agentic ai, 2026. URL https://arxiv.org/abs/2605.26112.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. Legomem: Modular procedural memory for multi-agent llm systems for workflow automation, 2025. URL https://arxiv.org/abs/2510.04851.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Andrej Karpathy. autoresearch: Ai agents running research on single-GPU nanochat training automatically. GitHub repository, March 2026. URL https://github.com/karpathy/autoresearch.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, Caiming Xiong, and Shafiq Joty. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems, 2025a. URL https://arxiv.org/abs/2504.09037.

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. MAS-ZERO: Designing multi-agent systems with zero supervision. *SEA@NeurIPS*, 2025b.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms, 2023. URL https://arxiv.org/abs/2304.08244.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Meng Chen, Congyu Zhang, Letian Zhang, Guiming Chen, Haoqin Tu, Xinyu Yang, Lu Feng, Xujiang Zhao, Haifeng Chen, Jiawei Zhou, Xiao Wang, Weitong Zhang, Hongtu Zhu, Yun Li, Jieru Mei, Hongliang Fei, Jiaheng Zhang, Linjie Li, Linjun Zhang, Yuyin Zhou, Sheng Wang, Caiming Xiong, James Zou, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Autoresearchclaw: Self-reinforcing autonomous research with human-ai collaboration, 2026a. URL https://arxiv.org/abs/2605.20025.

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, 2026b. URL https://arxiv.org/abs/2604.04323.

Sandeco Macedo. Stop hand-holding your coding agent: Engineering the loops that replace step-by-step prompting, 2026. URL https://arxiv.org/abs/2607.00038.

Shiva Krishna Reddy Malay, Shravan Nayak, Jishnu Sethumadhavan Nair, Sagar Davasam, Aman Tiwari, Sathwik Tejaswi Madhusudhan, Sridhar Krishna Nemala, Srinivas Sunkara, and Sai Rajeswar. Enterpriseops-gym: Environments and evaluations for stateful agentic planning and tool use in enterprise settings, 2026. URL https://arxiv.org/abs/2603.13594.

Yuzhen Mao and Azalia Mirhoseini. Decentralized multi-agent systems with shared context. *arXiv preprint arXiv:2606.10662*, 2026.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels?, 2025. URL https://arxiv.org/abs/2502.10517.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. Agentcl: Toward rigorous evaluation of continual learning in language agents, 2026. URL https://arxiv.org/abs/2606.02461.

Zachary S. Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. Core-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark, 2026. URL https://arxiv.org/abs/2409.11363.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. Paperbench: Evaluating ai's ability to replicate ai research, 2025. URL https://arxiv.org/abs/2504.01848.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, Weishu Zhang, Tyler Zeng, Ying Yan, Bo Liu, Hanson Wen, Mingyang Xu, Xiaoyuan Liu, Zimeng Chen, Weiyan Shi, Amanda Dsouza, Vincent Sunn Chen, Dawn Song, Patrick Bryant, Carl Boettiger, Yamini Rangan, Bradley Rothenberg, Kyle Steinfeld, Arvind Rao, Tapio Schneider, Georgios Yannakakis, Laure Zanna, Kaan Ozbay, Ida Sim, Tarek Zohdi, George Em Karniadakis, Jack Gallant, Teresa Head-gordon, et al. Agents' last exam. *arXiv preprint arXiv:2606.05405*, 2026.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383, 2024.

Hjalmar Wijk, Tao Roa Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Joshua M Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Jun Koba Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-bench: Evaluating frontier AI r&d capabilities of language model agents against human experts. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=3rB0bVU6z6.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation, 2025b. URL https://openreview.net/forum?id=z5uVAKwmjf.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen 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, 2026. URL https://arxiv.org/abs/2604.17308.

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, 2026. URL https://arxiv.org/abs/2604.20087.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering, 2026. URL https://arxiv.org/abs/2604.08224.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. Multiagentbench: Evaluating the collaboration and competition of llm agents, 2025. URL https://arxiv.org/abs/2503.01935.

Appendix

A Contributions **15**

B Extended Related Work **15**

C EvoHarnessBench Statistics **16**

D Metrics **19**

E Skill Construction Details **20**

E.1 Mining Reference Skills 20

E.2 Associating Skills with Tasks 21

F Agent Construction Details **21**

G System Details **23**

G.1 Single-agent Systems 23

G.2 Multi-Agent Systems 24

H Claude Code Results **24**

I Additional Analysis on Evolving Tools **25**

I.1 Single-Agent Systems 25

I.2 Multi-Agent Systems 26

J Additional Analysis on Evolving Skills **27**

J.1 Skill invocation details 27

J.2 Skill learning methods 28

K Additional Analysis on Evolving Agents **30**

A CONTRIBUTIONS

In summary, our key contributions are:

- We formalize **harness evolution** as a distinct source of non-stationarity in agentic systems. Unlike conventional continual evaluation, where tasks evolve under a fixed harness, we study settings in which the externally supplied tools, skills, or agents themselves change over time, reshaping what the system observes and what it can do.
- We introduce **EVOHARNESSBENCH**, a deterministic framework for constructing controlled, cumulative harness streams over *tools*, *skills*, and *agents* from verifier-based agent benchmarks. Capabilities are released from core to long tail, with tasks assigned to the earliest feasible stage such that each stage exercises newly introduced capabilities, while preserving the original tasks and deterministic evaluation.
- We evaluate systems under two complementary modes: *deployment evaluation*, measuring frontier-agent performance as the exposed action space expands, and *self-evolving adaptation evaluation*, measuring whether experience accumulated across harness versions supports adaptation and preservation or instead causes interference. The protocol captures adaptation, retention, transfer, and operating cost along the full harness trajectory.
- Through extensive evaluation of SAS and MAS, we reveal harness-induced forgetting, substantial variation across capability axes and environments, and persistent challenges in learning useful state under an expanding harness.

B EXTENDED RELATED WORK

Static Harness Benchmarks. Existing agent benchmarks evaluate increasingly rich harnesses, with tools, reusable skills, and collaborating agents representing three prominent forms of capability (Gu, 2026; Galster et al., 2026). Tool-use benchmarks evaluate whether agents can select, invoke, and compose executable interfaces (Yao et al., 2023; Li et al., 2023; Yao et al., 2024); skill-use benchmarks evaluate whether agents can correctly apply provided procedures, scripts, or references (Li et al., 2026; Liu et al., 2026b; Chen et al., 2026); and multi-agent benchmarks evaluate collaboration among predefined roles or specialist agents (Zhu et al., 2025; Liu et al., 2026a).

These benchmarks differ in the capabilities they expose, but share a common evaluation assumption: the harness is *fixed* within an evaluation condition. They measure how well an agent performs with a given set of tools, skills, or collaborators, rather than how its performance changes as that capability surface grows over time.

Benchmarks for Self-Evolving Agents. A growing line of work studies *self-evolving agents* that improve through repeated interactions by updating persistent components such as memory, skills, prompts, workflows, coordination strategies, or harness code (Ouyang et al., 2026; Zhang et al., 2025a; Hu et al., 2024; Zhang et al., 2025b; Ke et al., 2025b; Lee et al., 2026; Karpathy, 2026). These methods are commonly evaluated on challenging long-horizon scientific, software, and machine-learning benchmarks, including PaperBench, CORE-Bench, ScienceAgentBench, RE-Bench, MLE-bench, and KernelBench (Starace et al., 2025; Siegel et al., 2026; Chen et al., 2025; Wijk et al., 2025; Chan et al., 2025; Ouyang et al., 2025). Some self-evolution benchmarks, such as SkillLearnBench and SkillFlow (Zhong et al., 2026; Zhang et al., 2026), do involve a growing capability surface as agents generate and refine reusable skills over time. However, this growth is *agent-driven*—a product of the self-evolution process itself—and the evaluation targets skill generation quality rather than how task performance responds to that growth.

Across these benchmarks, the common pattern is that evaluation measures whether self-evolution improves final task performance, but does not track how the agent behaves across the evolution process—whether earlier competence is preserved, or whether accumulated artifacts remain useful as conditions change.

Continual Learning Benchmarks. Continual-agent benchmarks introduce temporal non-stationarity, but differ in *what* changes over time and *what* that change is used to evaluate. AgentCL constructs controlled task streams to measure transfer and memory-induced interference (Shu et al., 2026). ContinualLearningBench uses expert-validated task sequences with shared latent structure (Asawa et al., 2026). These benchmarks rigorously evaluate adaptation and preservation across

sequential tasks, but the agent's available harness remains fixed. Consequently, they can reveal when accumulated experience becomes harmful as the task distribution shifts, but not when that experience becomes stale because the harness through which the agent acts has changed.

The closest prior setting is CMRBench (Bell et al., 2026), which studies continual routing over an externally growing model pool. This is the only existing benchmark where harness growth is environment-driven rather than agent-driven. However, it considers a single axis (models) and evaluates routing decisions only, without end-to-end agentic execution or coordination. EVOHARNESSBENCH evaluates end-to-end agent performance under controlled harness growth across tools, skills, and agents.

C EVOHARNESSBENCH STATISTICS

This appendix reports construction statistics and diagnostics for each EVOHARNESSBENCH evolution axis. For every axis, we report both version-level statistics and construction checks. Version-level statistics characterize how the external harness grows over time; construction checks verify that each task is assigned to a valid release timestep and that the evaluated system receives the intended exposed harness rather than an oracle-pruned capability set.

Evolving Tools. We report tool-evolution statistics by version, including the number of newly released tools, cumulative tools, adaptation tasks, evaluation tasks, average oracle tools per task, and the fraction of exposed tools that are distractors for a given task. A distractor tool is any tool in the cumulative tool catalog $\mathcal{H}_t^{\text{tool}}$ that is not in the task's oracle tool set $\mathcal{C}_i^{\text{tool}}$. We report both the average number and fraction of distractor tools at each timestep.

In addition to aggregate statistics, we use four construction diagnostics. First, we verify that each retained task has a nonempty oracle tool annotation and a verifier-checkable outcome. Second, we verify that the tool catalog expands according to the release schedule and that $\mathcal{H}_t^{\text{tool}} \subseteq \mathcal{H}_{t+1}^{\text{tool}}$ for all t. Third, we check that the release schedule follows the intended core-to-long-tail structure by reporting tool frequency distributions before and after bucket assignment. Fourth, for each assigned task, we store the oracle tool configuration and the full exposed tool catalog at its assigned timestep. These records verify that the oracle tool set is covered by the exposed harness while ensuring that evaluated systems receive the full cumulative catalog, not an oracle-pruned tool list.

Evolving Skills. We report skill-evolution statistics by version, including task coverage, the number of newly released and cumulative latent skills, adaptation and evaluation tasks, essential-skill tags per task, and skill reuse frequency at each timestep. Task coverage is the fraction of seed tasks for which the deterministic skill matcher assigns at least one verifier-grounded essential-skill tag. Skill reuse frequency measures how many tasks are tagged with each latent skill and is used to construct the core-to-long-tail release schedule.

We also report diagnostics for the hidden reference skill construction. First, we report how much prompt content is retained as model-visible behavioral contract versus hidden as reusable procedural skill content. Second, we report the number of mined skill records and their entity, field, value, and outcome indices. Third, we report verifier-matching coverage: the number of tasks with at least one matched skill, the distribution of matched skills per task, and the number of tasks excluded because no deterministic match is found. Fourth, we verify that the hidden reference skill timeline is used only for task assignment and evaluation analysis: the evaluated system receives the stripped agent-facing prompt and seed-annotated tools, but not the hidden reference skills. These diagnostics ensure that the skill axis evaluates adaptive skill learning rather than direct retrieval of the reference skill curriculum.

Evolving Agents. We report agent-evolution statistics by version, including construction coverage, the number of newly released and cumulative agents, adaptation and evaluation tasks, agents per task, the fraction of tasks that require multiple agents, and the distractor-agent fraction at each timestep. A distractor agent is any specialist agent in the cumulative agent pool $\mathcal{H}_t^{\text{agent}}$ that is not in the task's annotated agent set $\mathcal{C}_i^{\text{agent}}$. Multi-agent task fraction measures the share of tasks whose oracle tools span at least two entity-scoped specialists.

We additionally report construction checks for the specialist-agent pool. First, we verify that every seed tool is assigned to exactly one owner entity by the ownership map $\eta : \mathcal{U}^{\text{tool}} \to \mathcal{E}$. Second, we verify that every active specialist has a nonempty tool bundle and that specialist tool bundles are

Table C.1: Tool-evolution statistics by domain and stage. Percentages are relative to each domain sample; adaptation and test splits are disjoint.

Domain	Stage	Train (adapt)	Test	Total tasks	% of sample	+ new tools	cumulative tools
EOG · Calendar	$\mathcal{H}_1$	12	29	41	67.2%	+17	17
	$\mathcal{H}_2$	4	10	14	23.0%	+7	24
	$\mathcal{H}_3$	2	4	6	9.8%	+4	28
	Total	**18**	**43**	**61**	**100%**	—	**28**
EOG · CSM	$\mathcal{H}_1$	5	13	18	17.5%	+25	25
	$\mathcal{H}_2$	7	15	22	21.4%	+10	35
	$\mathcal{H}_3$	9	20	29	28.2%	+11	46
	$\mathcal{H}_4$	10	24	34	33.0%	+15	61
	Total	**31**	**72**	**103**	**100%**	—	**61**
EOG · Drive	$\mathcal{H}_1$	13	29	42	65.6%	+22	22
	$\mathcal{H}_2$	4	10	14	21.9%	+6	28
	$\mathcal{H}_3$	2	6	8	12.5%	+8	36
	Total	**19**	**45**	**64**	**100%**	—	**36**
EOG · Email	$\mathcal{H}_1$	1	2	3	4.5%	+12	12
	$\mathcal{H}_2$	3	6	9	13.4%	+10	22
	$\mathcal{H}_3$	3	7	10	14.9%	+10	32
	$\mathcal{H}_4$	5	12	17	25.4%	+11	43
	$\mathcal{H}_5$	6	13	19	28.4%	+11	54
	$\mathcal{H}_6$	3	6	9	13.4%	+10	64
	Total	**21**	**46**	**67**	**100%**	—	**64**
EOG · HR	$\mathcal{H}_1$	4	10	14	13.7%	+23	23
	$\mathcal{H}_2$	5	13	18	17.6%	+13	36
	$\mathcal{H}_3$	11	25	36	35.3%	+12	48
	$\mathcal{H}_4$	9	22	31	30.4%	+14	62
	$\mathcal{H}_5$	1	2	3	2.9%	+4	66
	Total	**30**	**72**	**102**	**100%**	—	**66**
EOG · Hybrid	$\mathcal{H}_1$	10	24	34	38.6%	+82	82
	$\mathcal{H}_2$	5	11	16	18.2%	+25	107
	$\mathcal{H}_3$	4	9	13	14.8%	+25	132
	$\mathcal{H}_4$	8	17	25	28.4%	+31	163
	Total	**27**	**61**	**88**	**100%**	—	**163**
EOG · ITSM	$\mathcal{H}_1$	20	48	68	66.0%	+39	39
	$\mathcal{H}_2$	5	12	17	16.5%	+10	49
	$\mathcal{H}_3$	5	13	18	17.5%	+15	64
	Total	**30**	**73**	**103**	**100%**	—	**64**
EOG · Teams	$\mathcal{H}_1$	9	20	29	47.5%	+25	25
	$\mathcal{H}_2$	4	11	15	24.6%	+8	33
	$\mathcal{H}_3$	3	6	9	14.8%	+9	42
	$\mathcal{H}_4$	2	6	8	13.1%	+8	50
	Total	**18**	**43**	**61**	**100%**	—	**50**
ALE	$\mathcal{H}_1$	22	52	74	49.3%	+45	45
	$\mathcal{H}_2$	4	8	12	8.0%	+27	72
	$\mathcal{H}_3$	4	8	12	8.0%	+27	99
	$\mathcal{H}_4$	4	8	12	8.0%	+28	127
	$\mathcal{H}_5$	12	28	40	26.7%	+51	178
	Total	**46**	**104**	**150**	**100%**	—	**178**

pairwise disjoint. Third, we verify annotated-tool coverage: for every retained task, the union of the tool bundles owned by $\mathcal{C}_i^{\text{agent}}$ covers the task's oracle tool set $\mathcal{C}_i^{\text{tool}}$. Fourth, we report routing complexity through the number of specialists required per task and the fraction of tasks requiring coordination across multiple specialists. Finally, we verify the evaluation exposure: the lead agent receives the full cumulative specialist pool $\mathcal{H}_t^{\text{agent}}$ and no direct tool access, so success requires selecting, delegating to, and coordinating the appropriate specialists.

Table C.2: Skill-evolution statistics by domain and version. Percentages are relative to each domain sample; adaptation and test splits are disjoint.

Domain	Stage	Train (adapt)	Test	Total tasks	% of sample	+ new skills	cumulative skills
EOG · CSM	H_1	5	12	17	34.0%	+2	2
	H_2	5	13	18	36.0%	+2	4
	H_3	4	11	15	30.0%	+5	9
	Total	**14**	**36**	**50**	**100%**	—	**9**
EOG · HR	H_1	4	11	15	20.0%	+2	2
	H_2	9	22	31	41.3%	+2	4
	H_3	9	20	29	38.7%	+6	10
	Total	**22**	**53**	**75**	**100%**	—	**10**
EOG · ITSM	H_1	5	13	18	21.7%	+3	3
	H_2	8	20	28	33.7%	+2	5
	H_3	6	15	21	25.3%	+2	7
	H_4	5	11	16	19.3%	+3	10
	Total	**24**	**59**	**83**	**100%**	—	**10**
ALE	H_1	5	13	18	12.4%	+4	4
	H_2	4	11	15	10.3%	+2	6
	H_3	6	15	21	14.5%	+1	7
	H_4	11	25	36	24.8%	+2	9
	H_5	8	17	25	17.2%	+1	10
	H_6	9	21	30	20.7%	+3	13
	Total	**43**	**102**	**145**	**100%**	—	**13**

Table C.3: Agent-evolution statistics by domain and version. Percentages are relative to each domain sample; adaptation and test splits are disjoint.

Domain	Stage	Train (adapt)	Test	Total tasks	% of sample	+ new agents	cumulative agents
EOG · CSM	H_1	2	5	7	14.0%	+10	10
	H_2	2	6	8	16.0%	+3	13
	H_3	10	25	35	70.0%	+5	18
	Total	**14**	**36**	**50**	**100%**	—	**18**
EOG · HR	H_1	3	8	11	14.7%	+6	6
	H_2	2	13	15	20.0%	+3	9
	H_3	6	17	23	30.7%	+3	12
	H_4	11	15	26	34.7%	+6	18
	Total	**22**	**53**	**75**	**100%**	—	**18**
EOG · ITSM	H_1	4	7	11	13.3%	+4	4
	H_2	7	11	18	21.7%	+3	7
	H_3	3	12	15	18.1%	+3	10
	H_4	10	29	39	47.0%	+5	15
	Total	**24**	**59**	**83**	**100%**	—	**15**
ALE	H_1	12	24	36	24.0%	+2	2
	H_2	9	18	27	18.0%	+3	5
	H_3	5	17	22	14.7%	+3	8
	H_4	10	13	23	15.3%	+4	12
	H_5	4	17	21	14.0%	+4	16
	H_6	6	15	21	14.0%	+8	24
	Total	**46**	**104**	**150**	**100%**	—	**24**

In total, constructed from two verifier-based agent benchmarks, EnterpriseOps-Gym and Agents' Last Exam, EVOHARNESSBENCH comprises **802 unique benchmark tasks** instantiated as **1,510 axis-specific task examples** across 17 independent evolution trajectories (3–6 cumulative versions each; 70 harness snapshots). These timelines span **520 executable tools**, **42 latent reference skills**, and **62 specialist agents**. Skill construction covers 76.6% of eligible seed tasks, while 85.2% of agent-axis tasks require coordination across multiple specialists. By the final harness stages, each

Table C.4: Statistics of EVOHARNESSBENCH by harness axis and trajectory. Each row corresponds to one independent harness stream. *Sum across streams* sums the final harness sizes of individual trajectories, whereas *globally unique* deduplicates same-named capabilities across trajectories. *Req./Task* counts oracle tools, verifier-grounded reference skills, or annotated specialist agents, respectively. *Final-ver. distractors* is the mean fraction of the last version's cumulative capability pool that is non-oracle for a task.

Axis	Trajectory	Tasks	Stages	Final Harness Size	Req./Task	Final Distractors
Tools	Calendar	61	3	28	5.84	79.2%
	CSM	103	4	61	9.78	84.0%
	Drive	64	3	36	6.47	82.0%
	Email	67	6	64	5.69	91.1%
	HR	102	5	66	8.79	86.7%
	Hybrid	88	4	163	7.32	95.5%
	ITSM	103	3	64	6.93	89.2%
	Teams	61	4	50	8.23	83.5%
	ALE	150	5	178	3.11	98.3%
	Sum across streams	799	37	710	6.79	89.0%
	Globally unique	–	–	**520**	–	–
Skills	CSM	50	3	9	1.70	81.1%
	HR	75	3	10	2.95	70.5%
	ITSM	83	4	10	3.33	66.7%
	ALE	145	6	13	4.01	69.2%
	Sum across streams	353	16	42	3.30	–
	Globally unique	–	–	**42**	–	–
Agents	CSM	50	3	18	7.70	57.2%
	HR	75	4	18	5.63	68.7%
	ITSM	83	4	15	4.54	69.7%
	ALE	150	6	24	2.11	91.2%
	Sum across streams	358	17	75	4.19	76.8%
	Globally unique	–	–	**62**	–	–

task is evaluated alongside an average of 89.0% distractor tools or 76.8% distractor agents, creating increasing selection and coordination pressure as capabilities accumulate.

D METRICS

The core evaluation object is a **lower-triangular performance matrix** $P \in \mathbb{R}^{T\times T}$, where entry $P_{t,\tau}$ ($\tau \le t$) is the accuracy of the system on task cohort τ when evaluated under harness stage t:

$$P_{t,\tau} = \frac{1}{|\mathcal{D}_{\tau,\text{eval}}|} \sum_{(x_i,y_i)\in\mathcal{D}_{\tau,\text{eval}}} V(\mathcal{A}(x_i;\mathcal{H}_t,z_t),\ y_i)\,. \tag{1}$$

Each row corresponds to a harness stage; each column to a task cohort. The matrix is lower-triangular because cohort τ only exists once stage τ is reached.

Different views of this matrix answer different questions:

Adaptation (diagonal). $P_{t,t}$ measures accuracy on newly introduced tasks under their intended harness—can the system exploit newly available capabilities?

Retention (off-diagonal row). $P_{t,\tau}$ for $\tau < t$ measures accuracy on earlier task cohorts under the expanded harness—does the system preserve competence as capabilities accumulate?

Backward transfer (column delta).

$$\mathrm{BWT} = \frac{\sum_{i=1}^{T-1} n_i (P_{T,i} - P_{i,i})}{\sum_{i=1}^{T-1} n_i}, \tag{2}$$

where $n_i = |\mathcal{D}_{i,\mathrm{eval}}|$. BWT compares each cohort's final-stage performance against its performance at introduction. Negative BWT indicates forgetting: previously accessible competence has degraded as the harness expanded or as persistent state accumulated.

Forward transfer (diagonal delta).

$$\mathrm{FWT} = \frac{\sum_{i=2}^{T} n_i (P_{i,i} - P_{i,i}^{\mathrm{before}})}{\sum_{i=2}^{T} n_i}, \tag{3}$$

where $P_{i,i}^{\mathrm{before}}$ is cohort i's accuracy under harness $\mathcal{H}_i$ *before* stage-i adaptation (i.e., using z_{i-1}). FWT measures how much self-evolution at stage i improves performance on that stage's new tasks. Negative FWT indicates that adaptation actively harms the system's ability to use newly introduced capabilities.

Cumulative score. As an aggregate summary, we report micro-averaged accuracy across all cohorts at the final stage:

$$\mathrm{ACC} = \frac{1}{|\mathcal{D}_{\leq T,\mathrm{eval}}|} \sum_{(x_i,y_i)\in\mathcal{D}_{\leq T,\mathrm{eval}}} V(\mathcal{A}(x_i; \mathcal{H}_T, z_T),\ y_i)\,. \tag{4}$$

Cost. We additionally report operating cost (tokens, tool calls, latency) at each stage, capturing efficiency degradation as the harness grows.

Together, the matrix and its derived metrics diagnose *when* harness expansion helps or hurts (which stage), *what* is affected (new vs. old tasks), and *why* (harness-induced distraction vs. stale persistent state).

E Skill Construction Details

As described in the main text, we construct task-level skill annotations in two steps: first mining a deterministic reference-skill library, and then associating tasks with the reference skills related to their verifier-checked outcomes. Because EOG and ALE expose procedural information differently, the concrete implementation differs across the two benchmarks. No generative model is used in either construction.

E.1 Mining Reference Skills

EOG. EOG domains contain shared system prompts that mix general agent instructions with domain-specific procedural policies. We parse their section structure and retain general instructions—such as role definitions, behavioral constraints, and output requirements—in the agent-facing prompt, while extracting reusable procedural sections and subsections as reference skills. The extracted procedural content is removed from the prompt used in the skill track, so evaluated systems must access it through the provided skill library rather than through prompt-baked instructions. Domains whose prompts do not contain a sufficiently decomposable procedural policy are not included in the EOG skill track.

ALE. ALE does not contain an analogous shared procedural policy that can be separated from the task prompt. We therefore deterministically define a small catalog of reusable procedural atoms from recurring workflow and software requirements in the benchmark. Each atom consists of a canonical procedure together with fixed lexical patterns and, where applicable, software anchors. Cross-cutting instructions that apply to nearly all tasks remain visible as baseline instructions; the remaining atoms form the hidden reference-skill library. Unlike EOG, no task-specific content is removed from ALE prompts.

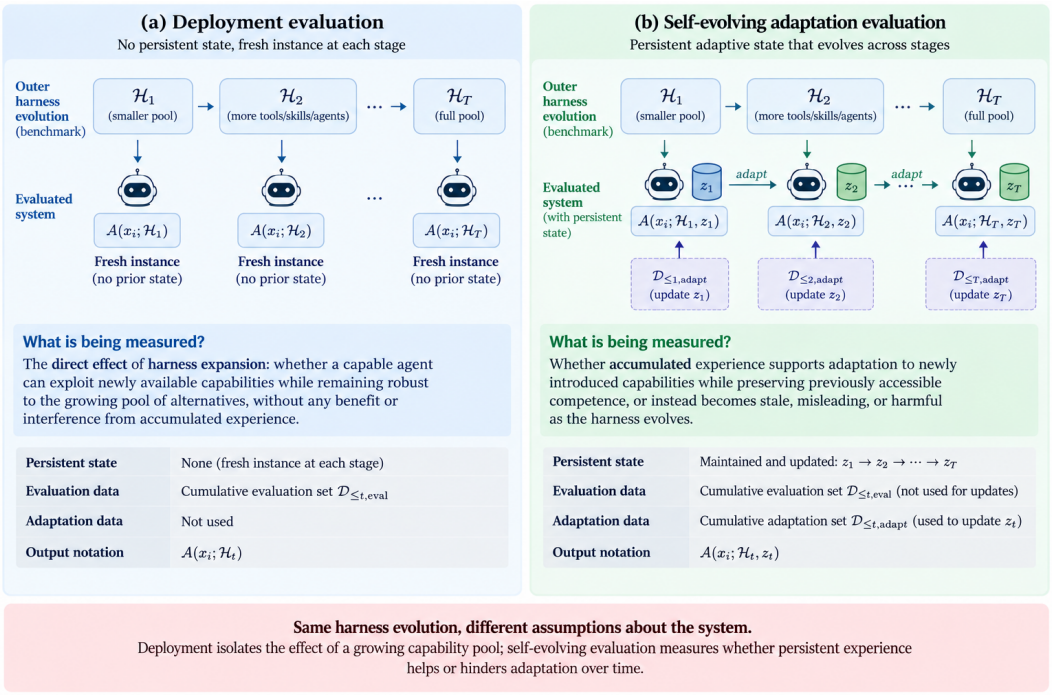


Figure D.1: Evaluation protocol of EVOHARNESSBENCH under the same outer harness evolution $\mathcal{H}_1 \subseteq \cdots \subseteq \mathcal{H}_T$. **(a) Deployment evaluation** uses a fresh system instance at each stage and evaluates it on the cumulative held-out task set, isolating the direct effect of harness expansion without accumulated experience. **(b) Self-evolving adaptation evaluation** maintains persistent adaptive state z_t, which is updated from adaptation data and carried across stages, and evaluates whether accumulated experience supports adaptation to newly introduced capabilities while preserving previously accessible competence.

E.2 ASSOCIATING SKILLS WITH TASKS

EOG. For each task, we parse its SQL verifiers to recover the table, column, and value constraints that determine successful execution, and extract the same identifiers from each reference skill. A skill is associated with a task when the verifier-checked state provides sufficient matching evidence: a value-level match is sufficient, while column-level evidence requires multiple matching constraints. System-managed fields such as identifiers and bookkeeping columns are ignored. The resulting matched set defines $\mathcal{C}_i^{\text{skill}}$.

ALE. Because ALE uses heterogeneous Python scorers rather than a common structured verifier language, association is lexical. A procedural atom is associated with a task when its fixed patterns match the task specification or when its software anchors overlap with the task's normalized required software. The resulting set again defines $\mathcal{C}_i^{\text{skill}}$.

Filtering and releases. Reference skills associated with no tasks and tasks with $\mathcal{C}_i^{\text{skill}} = \varnothing$ are excluded from the skill track. We retain only domains that support at least three non-empty harness stages. The remaining reference skills and task annotations are then passed to the shared frequency-ranked release construction in Section 3.2 to form $\mathcal{H}_1^{\text{skill}} \subseteq \cdots \subseteq \mathcal{H}_T^{\text{skill}}$.

F AGENT CONSTRUCTION DETAILS

The agent axis deterministically derives specialist agents from the capability structure already available in the source benchmarks. As described in the main text, we first map each tool-like capability to an owner and then convert each owner bundle into one specialist agent. No LLM or additional human annotation is used in this construction.

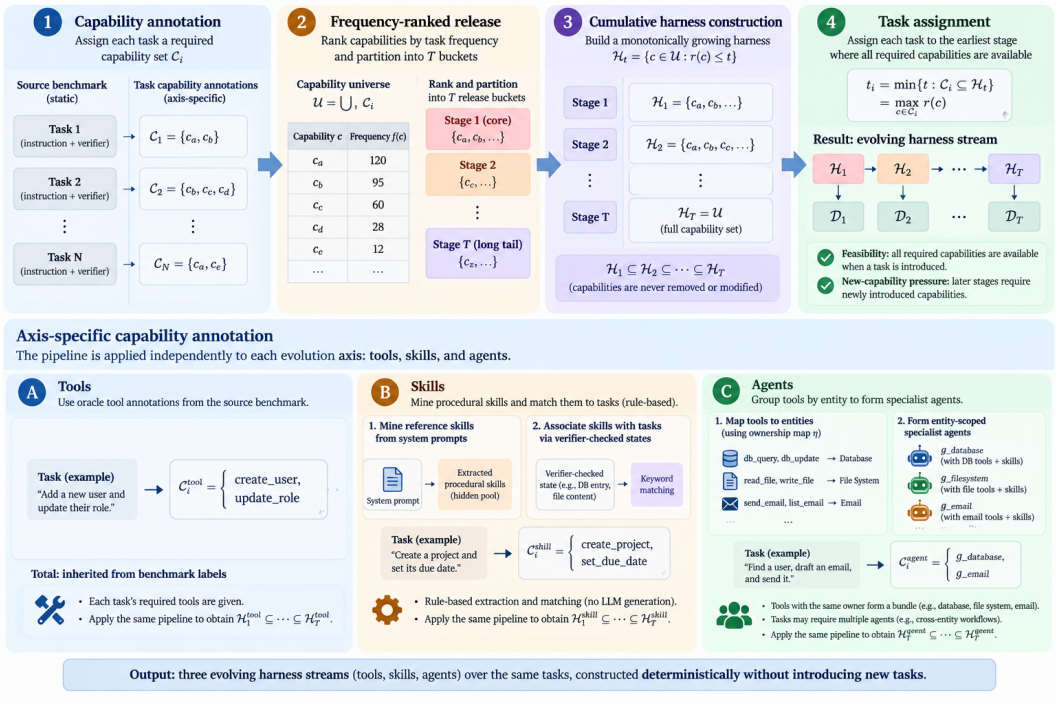


Figure E.1: Construction pipeline of EVOHARNESSBENCH. **Top:** Given task-level capability annotations, capabilities are ranked by task frequency, partitioned into stage-wise release buckets, and accumulated into a monotonic harness sequence $\mathcal{H}_1 \subseteq \cdots \subseteq \mathcal{H}_T$. Each task is assigned to the earliest stage at which all of its required capabilities are available, ensuring both feasibility and new-capability pressure. **Bottom:** Capability annotations are constructed independently for each evolution axis: tools inherit oracle annotations from the source benchmark, skills are mined and matched to tasks using rule-based procedures, and agents are formed by grouping tools into entity-scoped specialists. The same deterministic pipeline is then applied separately to obtain evolving tool, skill, and agent harness streams without introducing new tasks.

Constructing the ownership map. For EOG, tools operate on named entities in the underlying enterprise state (e.g., users, cases, incidents, or knowledge records). We recover the owner from the tool interface using a deterministic entity lookup, with more specific entity names matched before more general ones. Tools assigned to the same entity form one bundle. Thus, each EOG specialist corresponds to an entity and receives all tools associated with that entity.

ALE does not expose structured function tools; instead, tasks operate through software available in the execution environment. We therefore apply the same construction to the canonicalized software units produced by the tool axis, grouping related software into deterministic software families (e.g., Python libraries, command-line utilities, or domain-specific software stacks). Each family becomes one agent owner. In both benchmarks, every capability is assigned to exactly one owner, yielding a disjoint partition of the available capability universe.

Constructing specialist agents. For each owner e, we instantiate a specialist agent g_e with (i) a name and routing description derived from the owner, (ii) access restricted to the owner's tool or software bundle, and (iii) procedural context associated with that owner. The lead agent itself has no direct access to environment tools, so environment interaction must occur through the selected specialists. This ensures that when a task's required capabilities span multiple owners, successful execution requires delegation across the corresponding agents.

The procedural context is derived from the same deterministic skill-mining procedure used for the skill axis. For EOG, mined procedural content is associated with the entities to which the corresponding operations apply and attached to the relevant specialists. For ALE, software-anchored procedures are

attached to the corresponding software families, while cross-cutting execution instructions are shared across agents. No new procedural content is generated for the agent axis.

Task annotations and harness evolution. Given the resulting ownership map, a task's required agent set is obtained directly from the owners of its required capabilities, as defined in the main text. We then treat the constructed specialist agents as the capability universe and apply the same frequency-ranked release procedure from Section 3.2 to obtain the cumulative agent pools $H_1^{\text{agent}} \subseteq \cdots \subseteq H_T^{\text{agent}}$. Consequently, later stages expose the lead agent to both agents required by the current task and previously introduced agents that remain in the cumulative pool.

G System Details

G.1 Single-agent Systems

We evaluate several memory configurations to characterize how the content and representation of adaptation experience affect performance under evolving tools. Unless otherwise specified, memories are constructed from the adaptation split and provided to the agent cumulatively at each stage of the evolving environment. We consider both raw trajectory-based memories and compact memory representations that extract reusable information from prior interactions.

Raw (Simple Baseline) stores the complete interaction history from each prior adaptation episode. For every episode, the memory contains the task description, tool calls and their arguments, execution outcomes, and verifier feedback. These trajectories are retained without filtering, compression, or abstraction and are provided to the agent as stage-wise episodic memory. This configuration serves as the primary baseline for direct replay of prior experience.

Raw (task-specific tools in adapt. and eval.) uses the same raw episodic memory as **Raw**, but restricts the agent's action space to the ground-truth tools required for the current task. Thus, the agent still receives complete prior trajectories, while irrelevant tools are removed from consideration during execution. This configuration separates the effect of experience from the difficulty of selecting among an evolving set of tool candidates.

Raw (task-specific tools in adapt. only) further provides oracle access to the complete test-time tool configuration while retaining the raw adaptation memory. This condition removes both irrelevant tool-selection choices and uncertainty arising from the evolving test-time tool set, providing an upper-bound-style control for evaluating the contribution of memory under favorable tool availability.

Raw + Success Only retains only adaptation episodes that were successfully completed, discarding unsuccessful trajectories and their verifier feedback. The resulting memory therefore consists exclusively of successful demonstrations, while preserving the complete task descriptions and tool-use traces within those episodes. This configuration evaluates the effect of selectively retaining positive adaptation experience.

Raw (successful traj. only) retains only unsuccessful adaptation episodes, including their task descriptions, tool-use traces, execution outcomes, and verifier feedback. This configuration corresponds to `memory_filter=failure` and evaluates whether unsuccessful experience provides useful information for subsequent adaptation, despite not containing successful execution traces.

Reasoning Bank (Ouyang et al., 2026) replaces complete trajectory replay with compact reasoning artifacts extracted from prior adaptation episodes. Each episode is summarized into reusable reasoning items containing a title, description, and procedural reasoning derived from the observed interaction and outcome. Tool-use traces are not retained in their original form. This configuration therefore tests whether abstract procedural knowledge extracted from experience can provide more useful adaptation guidance than replaying complete trajectories.

Reasoning Bank (+ tool information) augments the reasoning artifacts with information about the tools encountered during adaptation. In addition to the abstract reasoning extracted from each episode, the memory retains the relevant tool identities and interface information needed to connect the reasoning to available actions. This condition evaluates whether explicitly preserving tool-related information improves the usefulness of otherwise abstract reasoning memories in evolving tool environments.

MemToolAgent (Er et al., 2026) represents each adaptation episode as a reflection rather than as a complete trajectory. The reflection summarizes the agent's experience, including relevant actions, environmental feedback, and task outcome, into concise procedural guidance that can be retrieved for subsequent tasks. This representation emphasizes reusable lessons from prior interactions while discarding execution-specific details that are preserved by raw trajectory memory.

G.2 Multi-Agent Systems

We provide detailed configurations of the multi-agent systems used in our evaluation. Since several methods were originally designed for less agentic settings, we make minor adaptations to fit our evaluation protocol while preserving their core architectures and design principles.

AutoGen (Wu et al., 2024). Under *evolving tools*, AutoGen uses a single solver agent that directly interacts with the available tools for up to 50 iterations. To mitigate repeated-action stalls, a secondary ground-truth role intervenes when the solver issues the same tool call twice consecutively. Under *evolving agents*, the solver instead acts purely as a router: it selects an appropriate specialist and delegates the task, after which the specialist executes independently in a fresh sub-conversation with a 12-iteration budget.

G-Memory (Zhang et al., 2025a). G-Memory follows the same execution structure as AutoGen (as its backbone), while augmenting it with persistent memory. Under *evolving tools*, the system stores task-level successes and failures in a hierarchical memory and retrieves relevant prior experience for subsequent tasks. Under *evolving agents*, the memory is adapted to record delegation decisions and their outcomes, thereby supporting specialist selection rather than direct tool use.

LegoMem (Han et al., 2025). LegoMem natively adopts an orchestrator-worker architecture. Under *evolving tools*, an orchestrator decomposes each task into 2-5 subproblems and assigns them to worker agents, each with a budget of 25 iterations. Workers share an intermediate workspace and retrieve memories of prior task decompositions and executions. By default, however, memories are maintained separately for each worker rather than shared across workers; cross-worker memory sharing is evaluated only as an ablation variant (*shared memory*) in Figure I.2. Under *evolving agents*, the same decomposition procedure is retained, but each subproblem is assigned directly to a named specialist from the current agent roster.

DeLM (Mao & Mirhoseini, 2026). DeLM uses an orchestrator-free, parallel execution scheme in which up to three workers independently process ready subproblems from a shared task pool, with at most three replanning rounds and 12 subproblems per task. By default, each agent maintains a raw memory that records its previous behavior, intermediate results, and outcomes. Under *evolving tools*, newly acquired information is verified against the originating worker's execution before being added to the shared workspace. Under *evolving agents*, each subproblem is additionally routed to a specific specialist, and the shared workspace records the provenance of specialist contributions.

H Claude Code Results

Claude Code is included as an additional frontier deployment system for the skill and agent axes. Because Claude Code differs from the controlled GPT-5/Codex systems in both its underlying model and native agent harness, we do not use its absolute performance for direct cross-system comparisons in the main tables. Its *within-system* comparison between task-specific and cumulative harnesses nevertheless provides an informative reference for how another frontier agent responds to harness evolution. We therefore report the complete aggregate results here and retain representative transfer results in the main text.

The within-system trends are consistent with the qualitative deployment findings in the main paper. Under evolving skills, cumulative exposure has little aggregate effect: Claude Code's EOG pass rate changes from 29.3% to 30.2%, while ALE remains at 13.2%. Under evolving agents, cumulative exposure has a modest negative effect, with pass rate changing from 11.0% to 10.6% on EOG and from 16.9% to 15.3% on ALE. These results provide an additional frontier-system reference for the conclusion that the effect of harness expansion is axis-dependent, while avoiding uncontrolled absolute comparisons with the GPT-5/Codex systems.

Table H.1: Claude Code deployment results under evolving skills and agents. Task-specific conditions expose only the capabilities annotated for each task, while cumulative conditions expose the full harness available at the corresponding stage. These results are intended for within-Claude-Code comparisons rather than direct comparison with the GPT-5/Codex-based systems in the main tables.

		EvoHarnessBench-EOG				EvoHarnessBench-ALE				Overall
Axis	**Harness**	Pass (%)	Score	(h)	Tok. (M)	Pass (%)	Score	(h)	Tok. (M)	**Pass Rate**
Skills	Task-specific	$29.3^{\pm 0.8}$	$65.5^{\pm 0.5}$	2.5	19.1	$13.2^{\pm 0.0}$	$35.0^{\pm 0.3}$	36.0	389.7	24.2
	Cumulative	$30.2^{\pm 1.4}$	$67.3^{\pm 1.0}$	3.8	13.6	$13.2^{\pm 1.2}$	$35.4^{\pm 1.5}$	36.7	422.7	24.8
Agents	Task-specific	$11.0^{\pm 1.7}$	$47.2^{\pm 0.7}$	10.3	7.1	$16.9^{\pm 2.0}$	$41.3^{\pm 1.4}$	32.4	240.3	12.8
	Cumulative	$10.6^{\pm 1.1}$	$49.5^{\pm 0.3}$	10.8	7.0	$15.3^{\pm 3.3}$	$42.3^{\pm 3.1}$	33.1	255.2	12.0

I Additional Analysis on Evolving Tools

I.1 Single-Agent Systems

In this section, we provide a breakdown analysis of memory-based adaptation under the *evolving tools* setting. Table I.1 compares memory variants that differ in the representation and content of retained experience, while Table I.2 examines how the adaptation harness itself affects the usefulness of accumulated memory.

Table I.1: Memory ablations for SAS under evolving tools (EvoHarnessBench-EOG). We compare raw memory with variants that selectively retain successful or failed experiences, incorporate oracle tools, or use alternative memory mechanisms.

	Pass (%)	Score
Raw Memory	**33.0**$^{\pm 0.9}$	**66.2**$^{\pm 0.6}$
Raw Memory + Oracle tool + full test	**35.3**$^{\pm 0.4}$	**67.7**$^{\pm 0.1}$
Raw mem + Oracle tool	**33.3**$^{\pm 0.4}$	**66.6**$^{\pm 0.6}$
Raw mem + Success only	**34.4**$^{\pm 1.7}$	**65.3**$^{\pm 0.8}$
Raw mem + Failures only	**34.4**$^{\pm 0.7}$	**66.1**$^{\pm 0.5}$
Reasoning Bank (Ouyang et al., 2026)	**36.9**$^{\pm 1.3}$	**68.7**$^{\pm 0.4}$
Reasoning Bank + Tools	**30.3**$^{\pm 1.0}$	**63.4**$^{\pm 0.5}$
MemToolAgent (Er et al., 2026)	**38.6**$^{\pm 1.0}$	**68.9**$^{\pm 0.5}$

The main results show that memory-based self-evolution accounts for several of the strongest gains under evolving tools. We therefore isolate how the *form and content of retained experience* affect adaptation. Rather than assuming that more memory is always beneficial, we ask which properties of prior experience provide useful signals as the tool catalog expands.

We ablate memory along three dimensions: *representation* (raw trajectories vs. structured reasoning or reflection), *content* (tool information and trajectory outcomes).

Structured experience is more effective than raw replay. Memory-based self-evolution benefits from transforming prior experience into structured representations. Raw trajectory memory improves pass rate from 30.2% to 33.0%, while Reasoning Bank and MemToolAgent achieve 36.9% and 38.6%, respectively. This suggests that the benefit of accumulated experience depends on how it is represented, rather than simply on retaining additional history.

More tool information is not necessarily better. Adding explicit tool information to Reasoning Bank substantially reduces performance from 36.9% to 30.3%. Thus, under an evolving tool catalog, exposing additional interface details within memory can introduce interference rather than improve adaptation. Other ablations on tool identity and trajectory outcomes show smaller differences.

As shown in Table I.2, Raw Memory adapted under the full cumulative tool catalog achieves 33.0% pass rate and 66.2 score on EOG. When adaptation instead uses task-specific tools, while evaluation remains under the same cumulative catalog, pass rate increases to 35.3%, score to 67.7, and adaptation time decreases from 25.7 to 20.5 hours. Because the evaluation harness is identical, this difference points to the adaptation context itself: experience grounded in task-relevant tools can produce more useful persistent state than experience acquired while reasoning over the broader cumulative tool space.

What Makes Past Experience Useful as the Tool Harness Evolves?

As shown in Table 2, memory-based self-evolving adaptation accounts for several of the strongest gains under evolving tools, with ReasoningBank and MemToolAgent among the best-performing methods. We therefore use memory as a concrete lens to study how the content and representation of past experience affect performance as the tool harness evolves. We ablate z_t along three dimensions: *content* (what is stored), *valence* (success vs. failure), and *fi-*

delity (real vs. corrupted tool identifiers). Our memory ablations (detailed in Table I.1) show that **how experience is represented matters more than simply retaining more experience**. Structured memory consistently outperforms raw trajectory replay: MemToolAgent achieves 38.6% pass rate compared with 33.0% for raw memory and 30.2% without memory. Moreover, adding explicit tool schemas to Reasoning Bank reduces performance from 36.9% to 30.3%, suggesting that additional tool information can introduce interference rather than improve adaptation. Finally, success-only and failure-only memories perform similarly (34.4%), indicating that unsuccessful interactions can provide useful signals for adapting to an evolving tool space. Together, these results suggest that effective self-evolving adaptation depends on **selectively extracting actionable experience**, rather than simply accumulating more context.

Table I.2: Effect of the adaptation harness for Raw Memory on EOG. All conditions follow the same task stream and are evaluated under the cumulative tool harness; "− broad exposure" uses only task-specific tool during adaptation;

Adapt. Harness	Eval. Harness	Pass (%)	Score	Time (h)
Cumulative	Cumulative	33.0	66.2	25.7
− broad exposure	Cumulative	**35.3**	**67.7**	**20.5**

I.2 Multi-Agent Systems

In this section, we provide a breakdown analysis of multi-agent systems and their memory variants under the *evolving tools* setting. Figure I.1 reports relative backward transfer (BWT) under both the *evolving tools* and *evolving agents* settings, while Figure I.2 compares different MAS memory variants under the *evolving tools* setting.

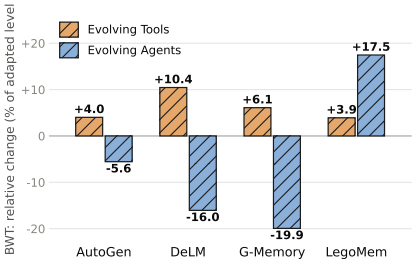


Figure I.1: Average Relative Backward Transfer (BWT) on EvoHarnessBench-EOG of MAS.

MAS exhibits little forgetting under evolving tools, but persistent memory provides no clear retention advantage over a no-memory baseline. For MAS, BWT remains uniformly small in magnitude, with all methods falling within ±2.3 percentage points. The no-memory AutoGen baseline, DeLM, and G-Memory show mild positive BWT (+2.3%, +1.2%, and +0.4%, respectively), while LegoMem is the only method with negative transfer (−1.7%). This suggests that, overall, performance on earlier task cohorts is largely preserved as the tool harness evolves. Moreover, LegoMem's degradation is concentrated in only two domains rather than broadly distributed, pointing to a mechanism-specific failure mode rather than a general limitation of memory-augmented adaptation.

Notably, however, none of the memory-based methods improves BWT over the no-memory AutoGen baseline, which achieves the highest value (+2.3%). Thus, unlike the SAS setting, persistent memory does not provide a clear retention advantage for MAS under evolving tools. This comparison should nevertheless be interpreted cautiously: high BWT alone does not imply effective continual adaptation, since a weaker baseline may appear stable simply because it acquires less new competence and therefore has less useful knowledge to forget. Overall, the small spread in BWT suggests that retention is not the primary differentiator among MAS memory mechanisms; their effectiveness must instead be assessed jointly with their ability to adapt to newly introduced tasks.

Memory can constrain or amplify search in multi-agent systems. Across EvoHarnessBench-EOG, memory exhibits strongly architecture-dependent effects under an evolving tool harness. **Most notably, adding memory does not consistently improve adaptation: its benefit depends more on what is stored and how it is used than on the mere presence of memory.** In AutoGen, concrete trajectory memory primarily sharpens tool selection: storing prior trajectories improves pass rate while reducing tool calls and failed verifications, and successful-only trajectories achieve the highest tool precision (77.4%) and exact-match rate (15.8%). In contrast, G-Memory's abstract summaries *encourage broader exploration, increasing calls and redundancy while lowering precision,* suggesting that abstraction alone does not effectively constrain an expanding tool space. LEGOMem shows the opposite pattern: its gains come not from more precise tool selection, but from broader execution and fewer failed verifications; removing high-level insights has little effect, whereas incorporating failed

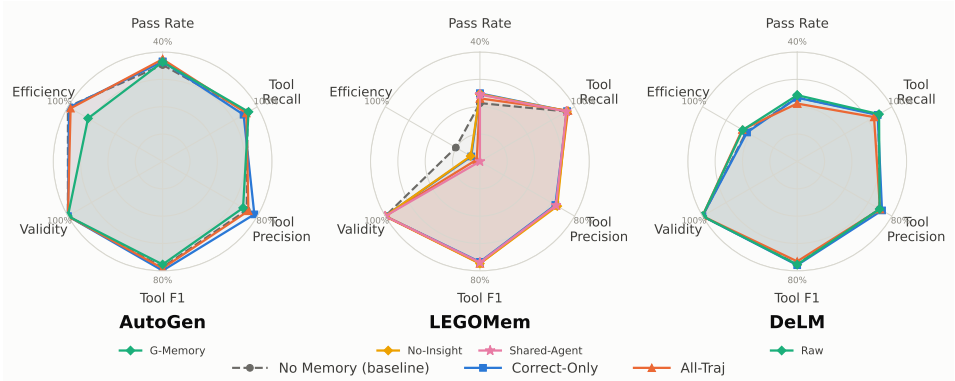


Figure I.2: Ablation study of MAS memory behaviors on EVOHARNESSBENCH-EOG. We focus on two commonly used components of agentic memory systems: (i) high-level insights abstracted from execution trajectories, and (ii) the choice of whether to retain only successful trajectories or to also incorporate failed trajectories into memory.

trajectories weakens the gain and sharing memory across sub-agents substantially increases execution cost. **DeLM provides the clearest counterexample to the assumption that richer memory is always better:** its default memory lowers pass rate by 2.2 points, while removing incorrect trajectories restores baseline performance and removing distillation yields the best result (+0.8 points). **Taken together, the strongest recurring signal is that concrete successful trajectories are consistently safer and more useful than abstracted, failed, or uniformly shared memories across sub-agents.** Thus, under harness evolution, multi-agent memory should be viewed not as a uniformly beneficial module, but as an architecture-sensitive mechanism whose effectiveness depends on memory quality, granularity, and placement.

SAS vs. MAS: Experience benefits depend on agent architecture. Self-evolution is architecture-dependent: accumulating experience consistently helps SAS, but its benefits become less predictable in MAS, suggesting that experience must be aligned not only with the task and harness, but also with how agents coordinate. In SAS, memory-based methods consistently improve over the deployment baseline, with MemToolAgent achieving the strongest overall performance (35.1%). In contrast, multi-agent systems show substantially more variable gains: G-Memory reaches 30.1%, while LegoMem achieves only 20.6%. These results suggest that experience that is effective when directly integrated by a single agent does not necessarily translate to multi-agent settings, where experience must be coordinated across agents and may interact with their individual decision processes.

J ADDITIONAL ANALYSIS ON EVOLVING SKILLS

J.1 SKILL INVOCATION DETAILS

Tool-oriented experience does not transfer cleanly to skill adaptation. Given that skill engagement is a central bottleneck above, we next ask whether memory mechanisms that are effective for evolving tools can help. Here, task-relevant skills are already provided, while the retained memories mainly encode tool-use and action experience rather than when or how to engage skills; accordingly, most memory variants remain close to the no-memory baseline. MemToolAgent is a stronger failure case: despite being the best memory method under evolving tools, its pass rate drops to 4.4% (Table J.1). This sharp degradation suggests negative transfer from tool-centric experience, potentially because the learned reflections bias execution toward tool-use patterns that are poorly aligned with the provided procedural skills. Together, these results suggest that persistent experience must be aligned with the type of capability being evolved.

Table J.1: Memory-based ablations under evolving skills (EVOHARNESSBENCH-EOG).

	Pass (%)	Score
Raw memory	17.7$^{\pm 6.8}$	55.2$^{\pm 4.4}$
Raw mem and oracle adapt and full test	23.9$^{\pm 1.4}$	60.4$^{\pm 0.3}$
Fake-tool memory	17.5$^{\pm 4.8}$	54.7$^{\pm 7.1}$
Reasoning-playbook memory	16.5$^{\pm 5.6}$	52.2$^{\pm 6.2}$
MemToolAgent (Er et al., 2026)	4.4$^{\pm 1.9}$	19.7$^{\pm 2.2}$

Skill invocation details. **(a)** Share of each system's offered skill library that is actually invoked (a distinct `SKILL.md` opened). **(b)** Precision and **(c)** recall of those invoked skills against the task-specific skill set. Bars are last-stage only (CSM v_3, HR v_3, ITSM v_4), pooled task-weighted over the three domains. GPT-5 unless labeled otherwise. Each bar uses that arm's own library as the denominator: the two task-specific Codex controls mount one task's skills, while Task-specific GEPA and the cumulative arms mount the stage library. Precision is omitted (*n/a*) when nothing is invoked. Claude Code's share counts file reads rather than distinct skills, and its precision/recall are omitted because skill names were not logged.

J.2 Skill learning methods

We describe the skill-learning methods that we evaluate as methods for adapting to an expanding skill-harness below:

Zero-shot. For each test task, the curator constructs a private skill overlay using only the task's visible instruction, its listed gold tool names, and the library snapshot available at that stage. The curator has no access to training tasks, training trajectories, or verifier outcomes. Thus, the resulting skills are conditioned on the current test task rather than distilled from prior solved experience.

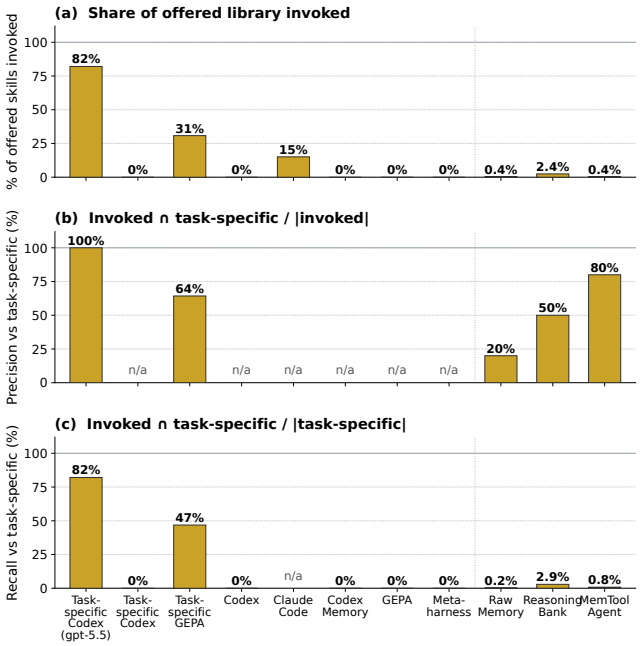


Figure J.1: Skill invocation and overlap with task-specific skills at the final stage.

Raw Traj. The skill library is initialized empty at h_1 and evolves throughout the curriculum. After the training trials at stage h_k, a batch curator processes the agents' compact trajectories—including messages, tool calls, and outputs—together with verifier pass/fail outcomes, and uses them to update a shared skill library. The resulting library is reused for test tasks at h_k and carried forward to h_{k+1}. Unlike the zero-shot setting, skills are derived directly from agents' execution experience rather than from gold skills or teacher-provided guidance.

Empty skills. This corresponds to an empty skill directory.

The other skill learning methods: One-shot, Self-feedback, Batch Self-feedback, Batch Teacher feedback and Skill-creator are from (Zhong et al., 2026).

Table J.2: Skill-learning ablations (Codex with GPT-5 on EvoHarnessBench-EOG). Methods vary in how skills are generated and refined from adaptation experience.

	Pass (%)	Score
Empty skills	$16.9^{\pm 2.5}$	$56.1^{\pm 1.4}$
Zero-shot	$17.6^{\pm 1.7}$	$54.5^{\pm 1.3}$
One-shot (Zhong et al., 2026)	$11.5^{\pm 0.6}$	$41.1^{\pm 0.4}$
Raw traj.	$20.0^{\pm 1.7}$	$59.3^{\pm 0.6}$
Self Feedback (Zhong et al., 2026)	$12.8^{\pm 2.2}$	$42.1^{\pm 0.8}$
Batch Self Feedback (Zhong et al., 2026)	$16.2^{\pm 2.4}$	$57.3^{\pm 2.5}$
Batch Teacher Feedback (Zhong et al., 2026)	$22.1^{\pm 3.0}$	$59.7^{\pm 0.8}$
Skill Creator (Zhong et al., 2026)	$20.9^{\pm 1.5}$	$60.6^{\pm 1.0}$

Skill learning: feedback quality determines skill quality.

- **Execution experience is necessary.** Skills extracted from raw trajectories (20.0%) substantially outperform both empty skills (16.9%) and test-conditioned zero-shot generation (17.6%). Without observing how tasks are actually solved, proposed skills lack grounding.
- **Iterative refinement with feedback matters most.** Batch Teacher Feedback (22.1%), Skill Creator (20.9%) are the strongest methods—all use some form of iterative refinement where skills are updated based on execution outcomes.
- **Naive skill learning fails.** One-shot generation (11.5%) and Self Feedback (12.8%) perform *worse* than having no skills at all (16.9%). Poorly generated skills actively mislead the agent—they are worse than nothing.
- **Batch processing helps.** Batch Self Feedback (16.2%) substantially outperforms single-episode Self Feedback (12.8%), suggesting that aggregating experience across multiple tasks before refining skills produces more robust procedures.
- **Overlap vs performance.** Figure J.2 shows that overlap with task-specific skills is only weakly associated with task performance ($r \approx 0.44$ for Pass and $r \approx 0.34$ for Score). In other words, closer alignment with the gold skill library does not consistently translate into better task performance, indicating that skill overlap captures only part of what matters for effective adaptation.

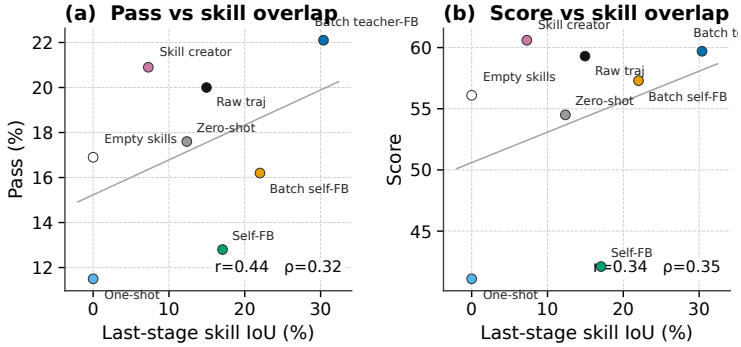


Figure J.2: **Relationship between last-stage skill overlap (IoU) and downstream performance** on EOG (Codex, GPT-5). Each point corresponds to a skill-learning method; IoU is averaged over CSM h_3, HR h_3, and ITSM h_4, while Pass and Score are taken from the final row of Table X. Error bars denote s.d. across domains for IoU and the reported s.d. for Pass/Score. r and ρ denote Pearson and Spearman correlations, respectively ($n = 7$);

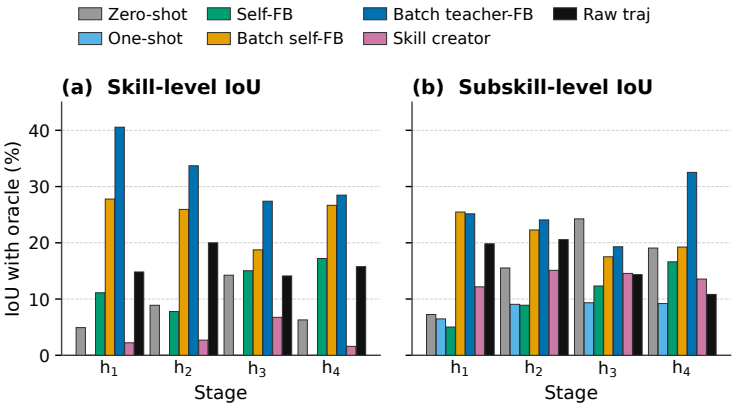


Figure J.3: Overlap of generated skills with the task-specific library across the curriculum. This is compute at stage-level since the skills are learned at every stage of the evolving-harness setup. The task-specific skills and sub-skills of all tasks in a stage are compared against the respective learned skills to get the iou.

Skill invocation is a model/agent property. At the final curriculum stage, skill invocation depends more on the underlying model and agent than on simply providing a skill library or memory store (Figure J.1). GPT-5-based Codex variants leave the offered skills entirely unread, while GPT-5.5 Codex invokes 82% of its task-specific skills. Task-specific GEPA also invokes skills (31% of those offered), though with lower precision (64%). Memory-based methods remain largely inactive,

invoking only 0.4–2.4% of the cumulative library. Thus, **providing or storing skills does not ensure their use: effective skill invocation is primarily an agent/model behavior.** .

K Additional Analysis on Evolving Agents

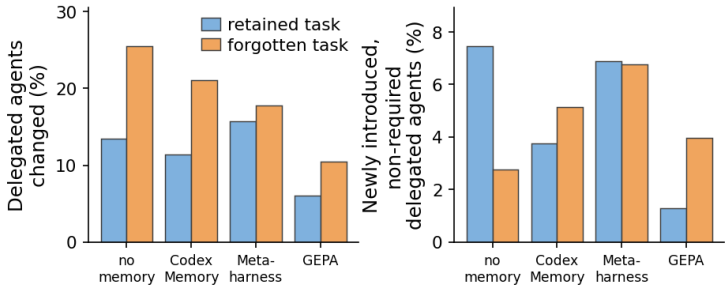


Figure K.1: Delegation drift on previously solved tasks on EvoHarnessBench-EOG. Left: change in the delegation set for retained vs. forgotten tasks. Right: share of later delegations to newly introduced, non-required agents.

Agent-pool expansion can break previously successful delegation patterns. To understand how evovling agent can disrupt previously solved tasks, we re-evaluate previously solved tasks under later agent pools and compare how delegation changes (Figure K.1). Forgotten tasks show substantially more delegation drift than retained tasks; without adaptation, for example, drift increases from 13% to 25%. This is not primarily caused by increased delegation to newly introduced, irrelevant agents. Instead, forgotten tasks more often stop invoking agents that were part of previously successful executions. Thus, harness-induced forgetting appears to arise mainly because agent-pool expansion disrupts effective delegation coverage. For newly introduced agents, the broader challenge is not simply reaching the new specialists, but maintaining coverage of required agents while avoiding unnecessary delegation as the pool grows (Figure K.5).

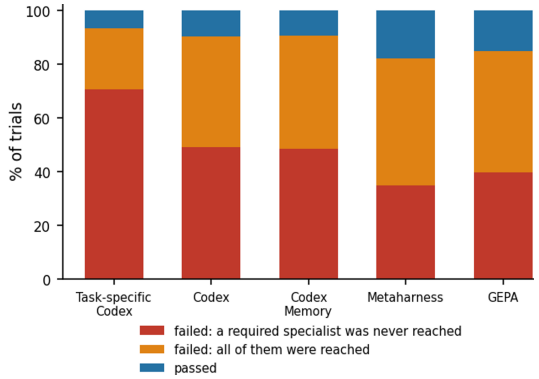


Figure K.2: Failure decomposition. Trials are divided into successes, failures that miss at least one required agent, and failures despite reaching all required agents.

Correct delegation is necessary but far from sufficient. Figure K.2 separates failures into those where at least one required specialist is never reached and those where all required specialists are reached but the task still fails. Self-evolution clearly reduces the first type: routing failures account for 54% of failures in the non-adaptive system, compared with 47% for GEPA and 43% for Meta-Harness. Yet the remaining coordination problem is substantial. Even conditional on reaching every required specialist, 78–86% of trials still fail their verifier. Thus, reaching the correct team is important but insufficient. The agent axis contains two distinct bottlenecks: *delegation*, identifying and reaching the relevant specialists under an expanding pool, and *coordination/execution*, successfully combining their work once the right specialists have been reached.

Forward transfer arises through different mechanisms. We next ask whether stage-specific adaptation teaches the orchestrator to use the specialists introduced at that stage. Figure K.3 compares

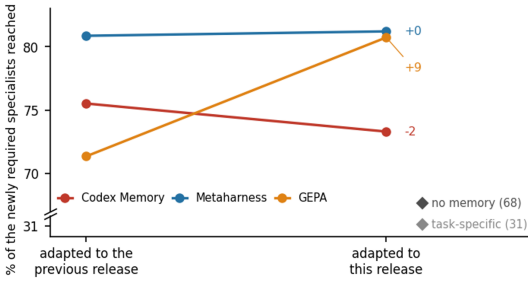


Figure K.3: Adoption of newly introduced agents.

new-specialist recall immediately before and after adaptation to each release. GEPA increases recall of newly required specialists from 71% to 81%, showing that its positive forward transfer is partly explained by learning to invoke newly exposed specialists. Meta-Harness, however, already reaches approximately 81% of the newly required specialists before adapting to the current release and shows essentially no further improvement after adaptation. Codex Memory slightly decreases. Positive FWT therefore does not have a single routing explanation: GEPA improves through better adoption of newly introduced specialists, whereas Meta-Harness's gains must arise downstream of specialist discovery, such as task decomposition, interaction between specialists, or execution after delegation.

Self-evolution adaptation improves delegation completeness, not selectivity. As shown in Figure 6, the main effect of self-evolution adaptation is not that the lead agent becomes more precise about which agents are relevant. Routing precision remains essentially unchanged at around 90% from the non-adaptive system to Codex Memory, GEPA, and Meta-Harness. Instead, required-specialist recall increases from 79% to 83–89%, while exact match rises from 27% to 33–38% (Figure 6, left). Importantly, these gains do not come from invoking more agents. As shown in the middle panel, GEPA reduces average delegations from 8.8 to 7.0 and repeated delegations from 3.8 to 1.4 while increasing required-agent recall to 88%. The right panel further shows that higher agent coverage corresponds to higher end-to-end success. Thus, self-evolution adaptation primarily helps the orchestrator *complete* the required delegation set and avoid redundant calls, rather than becoming more selective about which specialists are relevant.

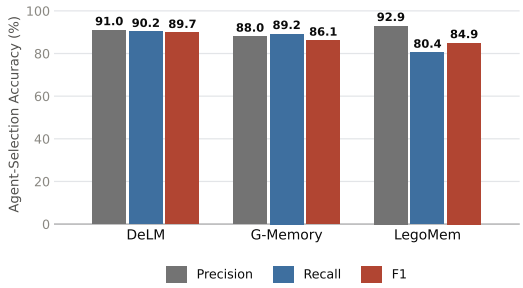


Figure K.4: Agent-selection precision, recall, and F1 by various MAS methods on the evolving-agents setting of EvoHarnessBench-EOG.

A similar pattern holds for MAS under an evolving agent pool. Across all evaluated methods, agent-selection quality is already high, with precision ranging from 88.0% to 94.2% and F1 from 84.9% to 90.2%, suggesting that identifying relevant agents is likewise not the primary bottleneck. More importantly, stronger selection accuracy does not consistently translate into better end-task performance. G-Memory, for example, has lower selection precision and F1 than DeLM, yet achieves a substantially higher final pass rate (18.2% vs. 14.2%). Conversely, LegoMem attains the highest selection precision (92.9%) while yielding the lowest final pass rate among the three memory-based methods. Together with the SAS results, this points to a consistent conclusion: once relevant specialists can be identified with reasonably high reliability, the dominant challenge shifts from *which* agents to select to *how* they are used—including task decomposition, routing, invocation, and coordination of their intermediate outputs. Self-evolution therefore appears to improve orchestration mainly by making delegation more complete and effective, rather than by materially improving selection precision itself.

Effect of newly-added sub-agents on MAS delegation. We further examine whether MAS routing mechanisms generalize to *newly introduced specialists* that have not appeared in earlier oracle labels, versus specialists already seen in prior versions. **The clearest finding is not that new specialists are uniformly harder to route to, but that routing degrades as the agent pool becomes increasingly cluttered with distractors.** At the individual domain–version level, old-specialist recall exceeds new-specialist recall in most transitions; however, after pooling by task count, this pattern persists only for G-Memory (90.6% vs. 86.8%), while DeLM (87.6% vs. 89.5%) and LegoMem (72.6% vs. 80.0%) reverse.

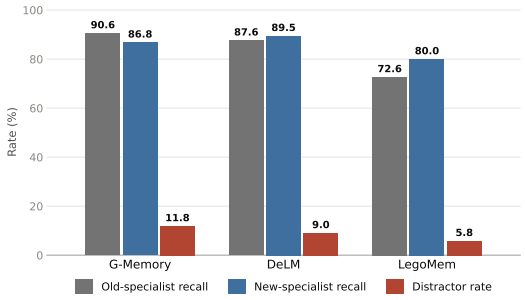


Figure K.5: Routing performance of MAS methods on old vs. newly introduced specialists, with distractor-selection rates.

Thus, the robust conclusion is that novel-specialist routing is locally harder in many transitions, but not consistently worse in aggregate. More importantly, the mechanisms expose a clear accuracy–caution trade-off: G-Memory attains the strongest specialist recall but also the highest distractor rate (11.8%), indicating aggressive over-delegation; LegoMem shows the opposite behavior, with the lowest distractor rate (5.8%) but substantially weaker recall, suggesting conservative under-delegation rather than precise routing; DeLM lies between these two extremes. **As the specialist pool grows, false delegation becomes the more consistent bottleneck:** for example, G-Memory's distractor rate on HR increases from 5.1% at v1 to 24.7% at v3. Overall, no mechanism simultaneously achieves both high specialist recall and low distractor routing, suggesting that the central challenge under agent evolution is not simply recognizing new specialists, but maintaining selective routing as the available agent pool expands.