HARNESSWEAVER: CONTINUAL SELF-IMPROVEMENT THROUGH TASK-CONDITIONED HARNESS COMPOSITION AND MODEL ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual self-improvement requires agents to use accumulated experience on subsequent tasks, yet failures may arise from limited executor capability or from inadequate planning, tool use, and execution control in the harness. We introduce HarnessWeaver, a two-timescale framework for continual self-improvement through task-conditioned model routing and harness composition without training model parameters. At task time, the Weaver uses accumulated evidence to select an executor and compatible components from a typed Living Pool, favoring low-cost configurations. Between rounds, it proposes model, harness, or joint revisions according to failure mechanisms and uses execution-based validation to admit persistent updates. On ALFWorld and Omni-MATH, frozen pre-adaptation references distinguish first-encounter transfer to new batches from recovery on repeated tasks. HarnessWeaver raises the equal-weight macro-average first-encounter success from 31.5% for the frozen reference to 56.8% (+25.3 points), outperforming the strongest external baseline, ACE at 39.1%, by 17.7 points. In independently rerun ALFWorld controls, Full reaches 58.6% first-encounter success, compared with 54.3% for Harness-only and 48.6% for Model-only, and also achieves the highest final-round success. These results show that reusable model–harness assemblies can serve as units of adaptation that connect accumulated experience, selective revision, and subsequent execution into a continual self-improvement process.

1 INTRODUCTION

Agent executions produce trajectories and feedback. Self-improvement requires turning these interactions into experience that improves performance on subsequent tasks. Prior work uses self-feedback to iteratively revise the current output within a task (Madaan et al., 2023). Distinct approaches to cross-task accumulation update model parameters from generated rationales or reward signals (Zelikman et al., 2022; Yuan et al., 2024), or retain experience as executable skills, context playbooks, or reasoning strategies (Wang et al., 2023; Zhang et al., 2026e; Ouyang et al., 2026). Other work uses execution traces to improve the agent harness (Chen et al., 2026). These lines of work respectively study within-task revision and effects on later tasks through model parameters or external execution state; they should not be conflated as the same form of experience accumulation.

Execution feedback, however, may reveal that a task failed without identifying whether the model or the harness should change. A failure may reflect insufficient executor capability, or inadequate planning, tool use, or execution control in the harness. Harness adaptation can narrow the performance gap between smaller and stronger models on some tasks, but its effectiveness still depends on the model's underlying capabilities (Yang et al., 2026). Routing models according to execution progress and cost can reduce workflow operating costs (Li et al., 2026), but does not by itself determine how the harness should be revised. As tasks continue to arrive with changing requirements, the system must therefore decide whether a failure calls for a different executor, a harness revision, or both. The scope of a revision matters as well: repeatedly applying dense updates to a single global harness can make it brittle on heterogeneous task streams (Liu et al., 2026).

We introduce HarnessWeaver, which treats the model–harness configuration as the object of task-level selection and continual adaptation. When a task arrives, the system uses task characteristics and

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.

historical evidence to select an executor and assemble compatible harness components, favoring low-cost configurations. After each round, it groups failures by mechanism and proposes model-only, harness-only, and joint revisions. Successful executions contribute reusable model–harness assemblies and performance evidence; failure-driven revisions must pass execution-based validation before they can be retained and reused within an appropriate scope. The system thus need not apply every local repair to all tasks or assume that every failure requires a stronger model.

Whether these updates constitute cross-task self-improvement cannot be determined from repeated attempts on previously encountered tasks alone. Prior work has noted that reporting final performance on the same task set used to search for a harness may overstate generalization to held-out tasks (Wang et al., 2026). Evaluating continual learning also requires examining whether experience transfers across tasks and whether irrelevant experience causes interference (Shu et al., 2026); observed gains on later tasks should be examined for evidence that they follow the intended save, retrieve, and update pathway (Xue et al., 2026). On ALFWorld and Omni-MATH, we distinguish first-encounter performance on subsequent tasks from recovery through repeated attempts, using frozen pre-adaptation baselines while tracking resource costs and checking source recovery and native-peer success during candidate validation. Full HarnessWeaver raises the equal-weight macro-average first-encounter success from 31.5% for the frozen reference to 56.8% and exceeds the strongest external baseline, ACE, by 17.7 points; independently rerun ALFWorld controls further compare Full, Harness-only, and Model-only adaptation surfaces.

Our main contributions are:

- We treat task-conditioned model–harness compositions as the unit of selection and adaptation, assembling sparse configurations from typed components instead of continually expanding a single global harness.
- We introduce an evidence-gated, two-timescale framework that favors cost-conscious task-level composition, compares model-only, harness-only, and joint revisions between rounds, and validates failure-driven updates before persistent admission.
- We develop a continual evaluation protocol with frozen pre-adaptation baselines to distinguish first-encounter transfer to later tasks from repeated-task recovery, while measuring resource cost and checking source recovery and native-peer success during candidate validation.

2 Related Work

Experience-based agent self-improvement. Agent experience can be incorporated into model parameters or retained outside the model for use on later tasks. STaR iteratively updates model parameters using generated rationales (Zelikman et al., 2022), while self-rewarding language models train with model-generated reward signals (Yuan et al., 2024). Non-parametric approaches retain experience in different forms: Reflexion uses verbal reflections to guide subsequent attempts (Shinn et al., 2023), and ExpeL extracts transferable natural-language insights from task experience (Zhao et al., 2024). Dynamic Cheatsheet retains strategies and code snippets across queries (Suzgun et al., 2026); ACE incrementally updates context playbooks (Zhang et al., 2026e); and ReasoningBank distills reasoning strategies from self-judged execution outcomes (Ouyang et al., 2026). These approaches primarily study how experience is generated, organized, and reused. HarnessWeaver additionally uses cases and performance evidence to select executors and harness components, and retains exact assemblies that succeeded in execution for later reuse.

Harness adaptation and model selection. Improvement can also target the system that organizes a model's execution. SICA explores edits to an agent implementation (Robeyns et al., 2025), the Darwin Gödel Machine iteratively modifies and empirically tests its own code (Zhang et al., 2026c), and Hyperagents makes the meta-level modification procedure itself editable (Zhang et al., 2026d). Among methods focused more directly on the harness, Self-Harness proposes small, trace-informed edits and tests them for regressions (Zhang et al., 2026b). HarnessX assembles typed harness primitives and adapts harnesses from execution traces through AEGIS (Chen et al., 2026); AutoSaddler combines failure diagnosis, structured patches, and validation-based selection in an offline optimization process (Park et al., 2026); and JIT-Agent trains a dedicated model to generate

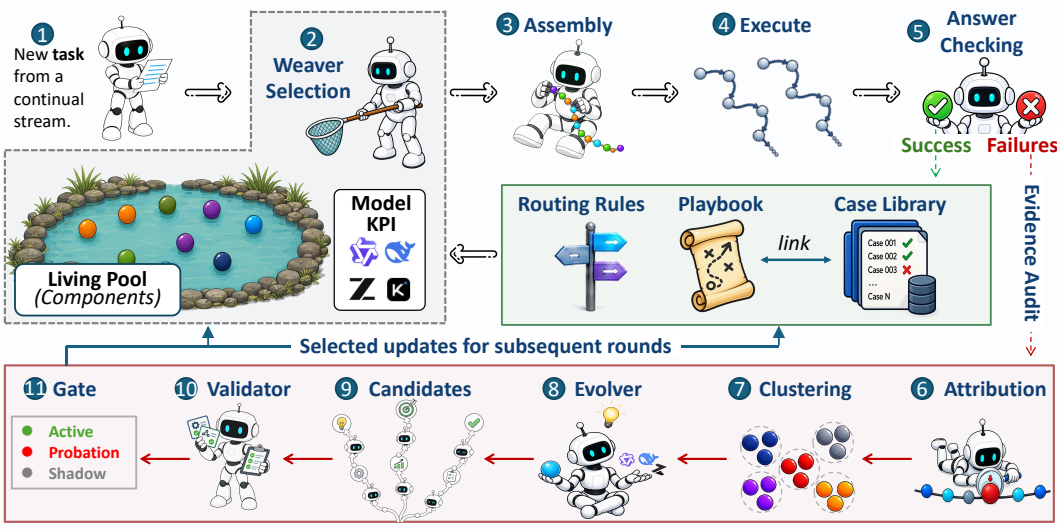

Figure 1: **Overview of HarnessWeaver.** The task-level loop (Steps 1–5) selects an executor and compatible harness components from the current state, executes the resulting assembly, and evaluates its outcome. Successful executions contribute reusable assemblies and performance evidence. The round-level loop (Steps 6–11) attributes and clusters eligible failures, proposes revisions, and validates them before tiered admission. Selected updates become available to subsequent rounds.

and repair task-adaptive harnesses (Zhang et al., 2026a). Adaptive Auto-Harness addresses open-ended task streams through a persistent harness tree and solve-time routing, enabling sustained harness construction and task-wise adaptation (Liu et al., 2026). Model selection provides a related line of work: Better Harnesses, Smaller Models shows that harness adaptation can narrow the gap between smaller and stronger models on some tasks, subject to the smaller model's underlying capabilities (Yang et al., 2026), while ProgRouter routes LLM agents according to workflow progress and time–cost budgets (Li et al., 2026). Building on these directions, HarnessWeaver treats executor selection and harness composition as coupled decisions and explicitly compares model-only, harness-only, and joint revisions for an identified failure mechanism.

Validation and evaluation of continual improvement. Repairing a source-task failure does not by itself establish that a revision will improve later tasks. Self-Harness and AutoSaddler already incorporate regression testing or validation-based selection into harness revision (Zhang et al., 2026b; Park et al., 2026). Work on evaluating harness evolution further notes that searching and reporting final performance on the same task set risks task-set-specific gains, motivating matched feedback and inference budgets and evaluation on held-out tasks (Wang et al., 2026). AgentCL uses controlled task streams to examine experience reuse and interference from irrelevant experience (Shu et al., 2026). PAST-Bench measures later-task gains alongside evidence that they follow the intended save, retrieve, and update pathway (Xue et al., 2026). Informed by these concerns, HarnessWeaver distinguishes source failures from native same-family tasks excluded from candidate generation during admission. Its continual evaluation separately reports first-encounter performance on new batches and recovery through repeated attempts on previously encountered tasks, against a frozen pre-adaptation reference, while recording resource cost and checking source recovery and native-peer success during candidate validation.

3 Method

3.1 Problem Setup and Overview

Let $\mathcal{X} = (x_1, \ldots, x_N)$ be a task stream in arrival order, $\mathcal{M}$ the available executor pool, and $B_1, B_2, \ldots$ its consecutive batches. Let $t \geq 0$ count completed adaptation rounds: H_0 is the initial state and H_t is used for execution round $t+1$. Each batch is executed for R rounds, with current batch index $j(t) = \lfloor t/R \rfloor + 1$. The state comprises components, routing rules, playbooks, cases, and performance statistics. At the start of each round, a read-only snapshot $\bar{H}_t$ is frozen so that all tasks

in the batch execute against a common state and their evidence can affect only the subsequent round. For $x_i \in B_{j(t)}$, the Weaver policy W selects $(m_i, A_i, \rho_i, p_i) = W(x_i, \phi(x_i), \bar{H}_t)$. Task-specific assembly and execution then follow

$$h_i := (m_i, \mathrm{Compile}(h_{\mathrm{base}}, A_i), \rho_i, p_i),\ \tau_i \sim \pi(h_i, x_i, \xi_i),\ y_i = V(x_i, \tau_i). \tag{1}$$

Here, m_i is the executor, A_i the set of type-compatible components, and ρ_i and p_i the optional matched routing rule and exact-assembly playbook; h_i is the compiled executable assembly. The features $\phi(x_i)$ are observable before execution and cannot uniquely identify an instance, ξ_i is the execution seed, and V is the verifier returning outcome y_i. The routing and playbook fields also record the assembly's provenance for audit.

We define round-level self-improvement as a recurrent update of the persistent system state. Let E_t denote the audited evidence from execution round $t+1$ and $E_t^{\mathrm{beh}} \subseteq E_t$ its reproducible behavioral failures. The validated and gated candidate updates are $\Delta_t = \mathrm{Gate}(\mathrm{Validate}(\mathrm{Evolve}(E_t^{\mathrm{beh}})))$; the subsequent state is

$$H_{t+1} = \mathrm{Commit}(\bar{H}_t, \mathrm{Learn}(E_t), \Delta_t). \tag{2}$$

where $\mathrm{Learn}(E_t)$ denotes experience updates to cases and KPIs together with exact-assembly playbooks induced by successful executions. KPIs are online success and resource statistics aggregated by task type for models and actually selected assets, and serve as evidence for later selection. Δ_t contains only model-only, harness-only, or joint candidates admitted by execution-based validation and the tiered Gate. This recurrence separates task-level execution from round-level adaptation and excludes unvalidated failure-derived changes from H_{t+1}. Figure 1 illustrates the resulting two-timescale loop.

3.2 Task-Conditioned Composition with a Living Pool

HarnessWeaver does not accumulate every revision in a single global harness. Instead, it maintains a task-reusable pool of typed components. The Living Pool consists of harness assets such as prompts, skills, processors, and tool controls, each with applicability conditions, compatibility constraints, execution evidence, and a lifecycle state. The Weaver may read the current task for semantic selection, but persistent rules and KPIs use only pre-execution features that cannot uniquely identify an instance; historical task text and full trajectories remain in the Case Library for audit and are not directly exposed to the Weaver. Before execution, the Weaver uses eligible assets and aggregated feedback to assemble a small compatible component subset with an executor. Compiled components govern context construction, action parsing, tool use, termination, and recovery. Subsequent real outcomes feed back into asset status and selection evidence. The decision-relevant persistent state after t adaptation rounds is represented as

$$H_t = (\mathcal{C}_t, \mathcal{R}_t, \mathcal{P}_t, \mathcal{Q}_t, \mathcal{S}_t, \mathcal{E}_t), \tag{3}$$

Here, $\mathcal{C}_t$ is the Living Pool. Its SHADOW, PROBATION, ACTIVE, and RETIRED states indicate a retained but non-deployable hypothesis, controlled trial, regular availability, and retirement, respectively. A SHADOW component remains in the pool but cannot enter a task assembly until promoted to PROBATION. $\mathcal{R}_t$ stores task-conditioned model-routing rules, $\mathcal{P}_t$ exact model–component playbooks, and $\mathcal{Q}_t$ candidate and validation records; these support the component pool but are not themselves components. $\mathcal{S}_t$ contains feedback statistics: model outcomes and resource use are aggregated by reusable task type, while actually selected components, rules, and playbooks accumulate usage evidence such as successes, tokens, and steps. $\mathcal{E}_t$ is the Case Library of real executions and outcomes. Valid executions update the corresponding statistics and cases; candidate validation and later use govern trial, promotion, and retirement. The Weaver consults deployable assets and aggregated evidence to reassemble a harness for each task rather than growing one global harness indefinitely.

Let $\ell_t(a)$ denote the lifecycle state of asset a in H_t, and let $\mathrm{compat}(a, x)$ indicate that its applicability conditions match the pre-execution features of task x. The initially eligible asset set for the current task is

$$\mathcal{A}_t(x) = \{a \in \mathcal{C}_t \cup \mathcal{R}_t \cup \mathcal{P}_t : \ell_t(a) \in \{\textsc{Active}, \textsc{Probation}\},\ \mathrm{compat}(a, x) = 1\}. \tag{4}$$

Equation (4) filters initially eligible assets by lifecycle state and task applicability. Final assembly additionally checks model availability, component conflicts and dependencies, and the component-count limit. This set and feedback statistics $\mathcal{S}_t$ in the frozen snapshot constrain the task assembly in

Equation (1). Thus $m_i \in \mathcal{M}_t$, $A_i \subseteq \mathcal{A}_t(x_i) \cap \mathcal{C}_t$, and $|A_i| \leq K$, where $\mathcal{M}_t \subseteq \mathcal{M}$ is the allowed model set and K the per-task component limit; any selected routing rule or playbook must also pass final assembly checks. Appendix B gives the full type contract.

Several components may be initially eligible for a task. To prioritize reliable components with lower context overhead rather than adding every component to the harness, the system computes the following selection weight for component a:

$$\omega_t(a) = A_t(a) R_t(a)\ \mathrm{clip}(1 + v_t(a), 0.5, 1.5) \left(1 + \frac{h_t(a)}{4000}\right)^{-1}, \tag{5}$$

Here A_t is the lifecycle factor, R_t is smoothed reliability from real use, v_t is separately stored validation marginal value, and h_t is marginal token overhead. The retrieval-based Weaver uses this weight to rank eligible components; the LLM Weaver used in the main experiment receives it in the component catalog as selection evidence rather than mechanically taking the highest score. The weight neither determines admission nor replaces the subsequent counterfactual Validator, and correlational KPIs are not interpreted as causal effects. Appendix B gives the smoothing and exploration constants.

3.3 Evidence-Driven Adaptation

Under the round-snapshot constraint of Section 3.1, execution evidence is audited first: infrastructure and protocol faults are quarantined, while only reproducible behavioral failures enter attribution. Attribution explains why an execution failed and clusters failures by shared mechanism, but does not prescribe what must change.

Evidence auditing partitions batch trajectories into disjoint success, behavioral-failure, and quarantine sets:

$$E_t = E_t^{\mathrm{succ}} \,\dot\cup\, E_t^{\mathrm{beh}} \,\dot\cup\, E_t^{\mathrm{quar}}, \qquad E_t^{\mathrm{beh}} = \{e \in E_t : y(e) < \theta,\ \mathrm{valid}(e) = 1,\ \mathrm{repro}(e) = 1\}. \tag{6}$$

This partition determines which experience may alter the system. E_t^{succ} supplies positive evidence for assemblies actually used, and an exact assembly from a normal successful execution can form a narrowly scoped ACTIVE playbook. E_t^{beh} supplies learnable failures to the evolution loop, while E_t^{quar} is retained only for diagnosis and reruns. Their disjointness prevents API, environment, or verifier faults from being misinterpreted as limitations of the model or harness. The attributor produces a failure mechanism z_i and confidence, after which a normalized mechanism key κ forms clusters

$$G_k = \{e_i \in E_t^{\mathrm{beh}} : \kappa(z_i) = k\}. \tag{7}$$

Here κ is formed from the diagnosed failure stage and mechanism, not task identity. Failures with the same stage and mechanism share a cluster G_k; incompatible failures remain separate.

For each eligible mechanism cluster, the Evolver proposes one candidate on each enabled intervention surface—executor-only, harness-only, or joint—and none on disabled surfaces. This separates diagnosis from intervention; the Evolver cannot directly create a playbook without validation. For G_k, the proposal space is

$$\begin{aligned} \mathcal{Q}_k = &\underbrace{\{(\delta_M, \varnothing) : \delta_M \in \mathcal{D}_M(G_k)\}}_{\text{model-only}} \\ &\cup \underbrace{\{(\varnothing, \delta_H) : \delta_H \in \mathcal{D}_H(G_k)\}}_{\text{harness-only}} \\ &\cup \underbrace{\{(\delta_M, \delta_H) : (\delta_M, \delta_H) \in \mathcal{D}_{MH}(G_k)\}}_{\text{joint}}. \end{aligned} \tag{8}$$

Here $\mathcal{D}_M(G_k)$, $\mathcal{D}_H(G_k)$, and $\mathcal{D}_{MH}(G_k)$ contain the modifications proposed on the respective surfaces: one for an enabled surface and none otherwise. For each $c \in \mathcal{Q}_k$, its evidence satisfies $\varnothing \neq \mathrm{evidence}(c) \subseteq \{\mathrm{id}(e) : e \in G_k\}$. A candidate may target the entire cluster or a coherent, explicitly scoped subpopulation; its evidence must cover every cluster task within the declared scope. All candidates face the same validation protocol, so attribution alone does not determine the adopted change. Appendix C.1 traces this pipeline concretely through an ALFWorld case in which a salt shaker is placed in the wrong drawer and recovery degenerates into repeated actions, from attribution and three-surface proposal generation through validation and persistent admission.

3.4 Counterfactual Validation and Persistent Update

The Validator runs candidate c on all cited, in-scope source failures with replayable incumbent records, $\mathcal{S}(c)$, and native same-family peers excluded from proposal generation, $\mathcal{P}(c)$; thus $\mathcal{V}(c) = \mathcal{S}(c) \cup \mathcal{P}(c)$. Source outcomes are compared with saved incumbent trajectories, while peers provide independent same-family task evidence.

Let y_i^0 be the immutable source outcome and y_i^c the candidate outcome. Source recovery and peer success are

$$R_S(c) = \frac{1}{|\mathcal{S}(c)|} \sum_{i \in \mathcal{S}(c)} \mathbb{I}[y_i^0 < \theta \wedge y_i^c \geq \theta], \qquad R_P(c) = \frac{1}{|\mathcal{P}(c)|} \sum_{j \in \mathcal{P}(c)} \mathbb{I}[y_j^c \geq \theta]. \tag{9}$$

R_P is defined only when $|\mathcal{P}(c)| > 0$; a candidate without peer evidence cannot enter ACTIVE. It measures peer success, not a transfer gain over the incumbent. Source comparisons use cached trajectories matched by task, incumbent model, components, seed, and rollout index, with the same verifier. This does not remove generation stochasticity or identify a causal effect; cross-batch transfer is evaluated separately against the frozen baseline.

The Gate first checks complete validation without infrastructure errors and the type, safety, and resource limits. Let $F(c)$ mean that these conditions hold, $N_S(c)$ be the number of recovered source tasks, and $\Delta_S(c) = |\mathcal{S}(c)|^{-1} \sum_{i \in \mathcal{S}(c)} (y_i^c - y_i^0)$. Initial admission is

$$g(c) = \begin{cases} \textsc{Active}, & R_S(c) = R_P(c) = 1,\ |\mathcal{P}(c)| > 0,\ F(c), \\ \textsc{Probation}, & N_S(c) \geq n_{\min},\ \Delta_S(c) > \delta_{\min},\ F(c), \\ \textsc{Shadow}, & \text{schema-valid but not admitted}, \\ \textsc{Reject}, & \text{otherwise}, \end{cases} \tag{10}$$

By default, $n_{\min} = 1$ and $\delta_{\min} = 0$. Without peers, positive source evidence can support PROBATION but not ACTIVE admission. With peers, at least one peer success provides peer-supported evidence; failure on every peer leaves only exploratory evidence. Schema-valid but unadmitted hypotheses may remain non-deployable SHADOW records. For the Gate-passing candidates $\mathcal{Q}_G^+$ with the same evidence scope, the winner is selected lexicographically:

$$\begin{aligned} c_G^* &= \operatorname*{arg\,min}^{\text{lex}}_{c \in \mathcal{Q}_G^+} \Psi(c), \\ \Psi(c) &= \big(r_{\text{adm}}, r_{\text{evid}}, -R_S, -R_P, -R_{\text{all}}, -\Delta_S, \\ &\qquad C_\$, C_{\text{GPU}}, L, T, K_{\text{int}}\big)(c). \end{aligned} \tag{11}$$

Here, $R_{\text{all}}(c)$ denotes the candidate's overall success rate over the validation set $\mathcal{V}(c) = \mathcal{S}(c) \cup \mathcal{P}(c)$, while r_{adm} and r_{evid} rank admission and evidence levels, respectively. Without peers, the ranking entry R_P is set to zero; this does not imply peer evidence. $C_\$$, C_{GPU}, L, T, and K_{int} denote monetary cost, GPU time, latency, tokens, and intervention complexity. Validation outcomes precede resource tie-breakers. Later real-task results update asset and assembly statistics and drive promotion, demotion, or retirement; Appendix A gives the full procedure.

3.5 Continual Evaluation Protocol

Batch j is first executed under $H_{(j-1)R}$ and last executed under H_{jR-1}. State H_{jR} is produced after that last execution and becomes available to the next batch. Scores refer to recorded execution rounds, not an additional evaluation of the post-update state.

We first freeze H_0 and execute every future batch B_j before any adaptation, obtaining $S(H_0, B_j)$. The frozen trajectories of B_1 are promoted directly as round-1 evidence, avoiding stochastic discrepancies from redundant execution; evolution then continues on B_1 through round 5. Round 6 is the first execution of B_2 under the evolved system. Since Round 1 reuses frozen trajectories, $\Delta_{\text{FE}}^{(1)} = S(H_0, B_1) - S(H_0, B_1) = 0$ by design. Evidence of cross-batch transfer therefore begins with Batch 2; subsequent rounds on each batch measure within-batch recovery.

For batch j, first-encounter transfer gain is

$$\Delta_{\text{FE}}^{(j)} = S(H_{(j-1)R}, B_j) - S(H_0, B_j). \tag{12}$$

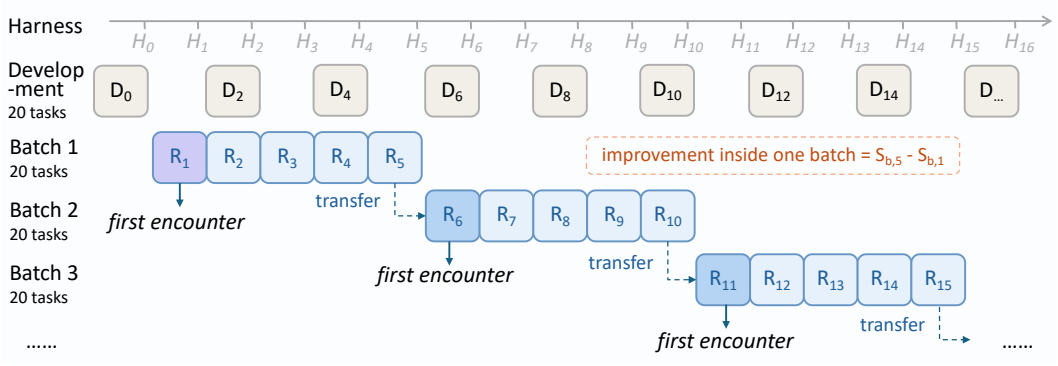

Figure 2: Continual evaluation schedule with a frozen baseline, first encounters, and within-batch rounds. Frozen H_0 evaluates development and every future batch before adaptation. Batch 1 reuses its frozen trajectory as Round 1; later batches begin with a first encounter under the evolved state. Read-only development evaluation occurs every two rounds.

Within-batch gain is $\Delta_{\mathrm{WB}}^{(j)} = S(H_{jR-1}, B_j) - S(H_{(j-1)R}, B_j)$. The overall prequential metric counts only the first time each task is encountered by the online system; later repetitions are reported separately as learning curves and are not conflated with generalization. Let $\mathcal{D}$ be a fixed development set disjoint from both the evolution stream and peer bank, and write $D_t = S(H_t, \mathcal{D})$. We measure D_0 before adaptation and then evaluate $D_2, D_4, \ldots$ on the latest committed state every two rounds. These scores diagnose whether persistent updates transfer beyond the active batch and reveal performance regressions. They neither contribute to the primary metric nor drive proposal generation, validation, the Gate, checkpoint selection, or hyperparameter selection, and they are never written back to cases, KPIs, routes, components, or peer retrieval.

4 Experiments and Results

4.1 Experimental Setup

Tasks and evaluation. We evaluate HarnessWeaver on embodied textual interaction using the ALFWorld valid-unseen split (Shridhar et al., 2020), and on mathematical reasoning using the legacy 478/100 split distributed in an early EvoAgentBench-compatible release, derived from Omni-MATH (Gao et al., 2025; 2026). ALFWorld is scored by environment-goal completion, whereas Omni-MATH uses a pinned Omni-Judge checkpoint. Table 1 summarizes dataset sizes and batching. Section 3.5 and Figure 2 define the evaluation protocol; Appendix D gives the detailed splits and implementation settings.

Table 1: Evaluation protocol and data splits.

Benchmark	Evolution	Batch plan	Rounds/batch	Development	Peers
ALFWorld	70	20/20/20/10	5	20	44
Omni-MATH	100	5×20	5	20	458

Models. The default task executor is Qwen3.5-9B, and the Weaver uses Qwen3.8-27B. The selectable executor pool also includes models such as GLM-5.3-Flash, Qwen3.8-Flash, and DeepSeek-V4-Flash. Appendix Tables 4, 5, and 3 give the complete model pool, role-specific generation settings, and runtime protocol, respectively.

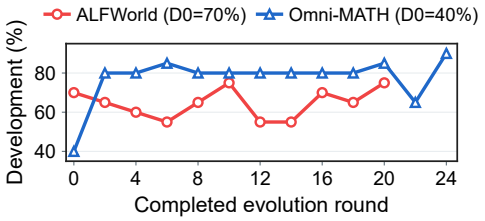

Figure 3: Fixed-development success; legend D_0 values are pre-adaptation references.

4.2 Main Results

Table 2 compares the frozen reference, single-pass external baselines, and the HarnessWeaver family. Baseline comparison uses first-encounter success over the complete stream, counting each task once. Continual methods additionally report task-weighted final-round success and its change from first encounter. Macro averages weight the two benchmarks equally. Full HarnessWeaver delivers substantial gains on both benchmarks. On ALFWorld, first-encounter success rises from 50.0%

Table 2: Success rates (%). Δ_{FE} is first encounter minus the shared frozen reference in the first row for each benchmark; Δ_{WB} is final minus first encounter. For frozen and single-pass methods, Final equals First and $\Delta_{\mathrm{WB}} = 0$ by protocol. Green/red denotes positive/negative change; best and second-best values are bolded and underlined.

	ALFWorld				Omni-MATH				Macro avg.			
Method	First	Δ_{FE}	Final	Δ_{WB}	First	Δ_{FE}	Final	Δ_{WB}	First	Δ_{FE}	Final	Δ_{WB}
Frozen reference												
Qwen3.5-9B + Initial Weaver	50.0	0.0	50.0	0.0	13.0	0.0	13.0	0.0	31.5	0.0	31.5	0.0
External baselines												
DC (Suzgun et al., 2026)	55.7	+5.7	55.7	0.0	7.0	−6.0	7.0	0.0	31.4	−0.1	31.4	0.0
ACE (Zhang et al., 2026e)	67.1	+17.1	67.1	0.0	11.0	−2.0	11.0	0.0	39.1	+7.6	39.1	0.0
HarnessWeaver family (full + cost-oriented variants)												
Full HarnessWeaver	58.6	+8.6	81.4	+22.9	**55.0**	**+42.0**	**65.0**	+10.0	56.8	+25.3	73.2	+16.4
w/o Native Peer	**72.9**	**+22.9**	82.9	+10.0	45.0	+32.0	60.0	+15.0	**58.9**	**+27.4**	71.4	+12.5
w/o Lifecycle	61.4	+11.4	**88.6**	**+27.1**	54.0	+41.0	64.0	+10.0	57.7	+26.2	**76.3**	**+18.6**
w/o KPI Feedback	62.9	+12.9	82.9	+20.0	42.0	+29.0	54.0	+12.0	52.4	+20.9	68.4	+16.0
Accept-All	71.4	+21.4	67.1	−4.3	44.0	+31.0	60.0	**+16.0**	57.7	+26.2	63.6	+5.9

under frozen H_0 to 58.6% (+8.6 points), and task-weighted final-round success further reaches 81.4% (+22.9 points). Figure 4 shows that Batches 2–4 improve over their respective frozen references by 5, 15, and 20 points at first encounter and finish at 75%, 90%, and 70%. On Omni-MATH, first-encounter success increases from 13.0% to 55.0% (+42.0 points) and reaches 65.0% on final rounds; Appendix Figure 7 shows first-encounter gains of 60, 75, 30, and 45 points in Batches 2–5, respectively. Its adaptation is primarily routing-driven: Qwen3.7-Flash accounts for 446 of 500 online executions, while all four seed mathematical components remain reusable. Relative to the baselines, the full method establishes a clear advantage on Omni-MATH and, through continual adaptation, reaches a higher ALFWorld final-round result than the strongest single-pass baseline. Within the HarnessWeaver family, Full serves as the complete reference configuration, retaining native-peer validation, lifecycle management, KPI feedback, and the counterfactual performance Gate; the lightweight variants selectively remove one mechanism to characterize the trade-offs among effectiveness, cost, and control strength. Section 4.4 provides the corresponding analysis.

Results on the fixed development sets in Figure 3 further show that persistent updates transfer beyond the active evolution batch. ALFWorld starts from $D_0 = 70\%$ and finishes at 75%. Omni-MATH rises rapidly from 40%, remains near 80% at most checkpoints, and reaches 90% at Round 24. These results align with the gains on the primary stream and indicate that HarnessWeaver's learned updates transfer across tasks.

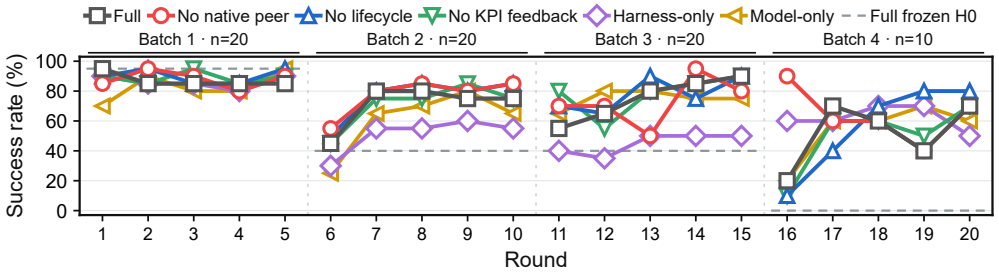

Figure 4: Round-wise ALFWorld success across six configurations. Gray dashes mark Full's frozen reference; each batch starts with first encounter, followed by repeated evaluations of the same tasks.

4.3 Adaptation-Surface Ablations

We isolate the object of adaptation with two independently rerun internal controls. Harness-only fixes the executor and permits only harness updates, whereas Model-only freezes the initial component set and permits only model-routing updates. Figure 5 shows first-encounter success

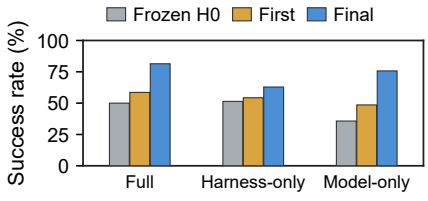

Figure 5: ALFWorld ablation results.

of 54.3% and 48.6%, respectively, compared with 58.6% for full HarnessWeaver in Table 2. The 4.3-point gap between Harness-only and Full is smaller than the 10.0-point gap for Model-only, while Full also achieves the highest final-round success (81.4%, versus 62.9% and 75.7%, respectively). Full leads on both metrics in these runs, supporting the design choice to retain both adaptation capabilities; the figure displays each run's own frozen starting point to contextualize the comparison. Appendix G.2 specifies the mutable state and retained protocol of each control.

4.4 Cost and Efficiency

The four lightweight variants remove native-peer validation, lifecycle transitions, KPI feedback, or the counterfactual performance Gate; Appendix G.3 details each intervention. Figure 6 separates task-execution from evolution-and-validation tokens, with savings reaching 34% on ALFWorld and approximately 10% on Omni-MATH. Some relaxed variants attain higher individual endpoints, but they do so under weaker evidence or state controls; Accept-All, for example, loses 4.3 points from first encounter to the final ALFWorld round. Exact counts appear in Appendix Table 12.

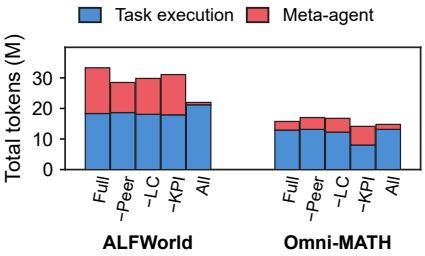

Figure 6: Online tokens (M): execution (blue), evolution and replay (red). All = Accept-All.

4.5 Choosing Model, Harness, or Joint Revisions

Harness changes target procedural failures, such as repeated search, omitted subgoals, and unverified termination on ALFWorld, or premature submission, malformed answers, and omitted answer checks on Omni-MATH. Model changes target capability bottlenecks that persist despite adequate procedural support, such as recurring failures in multi-step action planning or key mathematical derivations; these motivate testing a better-suited executor rather than adding further procedural constraints.

Joint changes should be compared when procedural and reasoning difficulties interact: the harness supplies search, subgoal, or answer-checking support, while the executor performs the corresponding planning and derivation. When neither single-surface revision sufficiently recovers the tasks, but a joint candidate provides stronger evidence on the same source tasks and peers while satisfying resource constraints, the system selects both changes. Logged ALFWorld trial selections include 13 routing, 7 harness, and 3 joint selections, confirming that all three surfaces are used in practice. Asset-count and lifecycle statistics are provided in Appendix D.3.

5 Conclusion

We introduced HarnessWeaver, a framework for non-parametric self-evolution through a continually maintained pool of task-conditioned executable harnesses. It quarantines engineering faults, attributes behavioral failures by mechanism, proposes model, harness, and joint interventions, and governs persistent updates using execution evidence from source tasks and native peers under explicit resource constraints. In the completed runs, ALFWorld first-encounter success improves by 8.6 points over the frozen baseline and reaches 81.4% on task-weighted final rounds; Omni-MATH rises from 13.0% to 55.0% at first encounter and reaches 65.0% on final rounds, demonstrating substantial cross-task transfer and within-batch recovery from evidence-constrained modular adaptation.

Limitations

The present evidence is limited to a small set of benchmarks and single completed runs, and repeated within-batch evaluation may retain task-specific experience; the method also depends on language-model-generated attribution and proposals, the quality of benchmark-provided peers, and the expressivity of a predefined component language and frozen model pool. Omni-MATH lacks complete Harness-only and Model-only controls, so its gains do not isolate each adaptation surface. Structured checks, counterfactual execution validation, and lifecycle management mitigate these risks, while future work should test broader task streams, independent replications, and richer but still auditable adaptation spaces.

AI USE STATEMENT

Generative AI tools were used both for research ideation or execution and to aid or polish writing. Specifically, they assisted discussion of research questions and experimental design, software implementation and debugging, audit of logs and results, preparation of scientific figures, and drafting, translation, and readability editing of portions of the manuscript. The studied system itself also uses language models in the Weaver, attribution, Evolver, and validation-related roles; the corresponding model roles and protocols are documented in the main text and appendix. The authors reviewed all AI-assisted content, checked numerical results and figures against the original run records, and verified the associated code, claims, and literature references. Generative AI was not credited as an author and was not used to generate benchmark ground truth or alter experimental observations. The authors take full responsibility for the final text, claims, code, and artifacts.

ETHICS STATEMENT

This work follows the ICLR Code of Ethics. All experiments use the publicly available ALFWorld and Omni-MATH benchmarks. The study involves no new human-subject research and does not collect or release new personal data.

REPRODUCIBILITY STATEMENT

The main text specifies the method, continual evaluation protocol, experimental setup, and metric definitions. The appendix further documents the algorithm, implementation contract, model roles, data partitions, candidate-admission and lifecycle rules, and validity and exclusion criteria. The anonymous supplementary material provides the implementation, experiment configurations, task manifests, and instructions for running the ALFWorld and Omni-MATH protocols. Together, these materials document the task ordering, model roles, admission rules, lifecycle settings, and task-weighted aggregation used in the reported experiments.

REFERENCES

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Wang Benyou, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *International Conference on Learning Representations*, volume 2025, pp. 100540–100569, 2025.

Xingze Gao, Chuanrui Hu, Hongda Chen, Pengfei Yao, Zhao Wang, Yi Bai, Zhengwei Wu, Yunyun Han, Xiaofeng Cong, Jie Gui, et al. Evoagentbench: Benchmarking agent self-evolution via ability transfer. *arXiv preprint arXiv:2607.05202*, 2026.

Songyuan Li, Ahmed M Abdelmoniem, and Shiqiang Wang. Progrouter: Online progress-guided orchestration for multi-agent llm workflows under quality-cost tradeoffs. *arXiv preprint arXiv:2608.25992*, 2026.

Zewen Liu, Zhan Shi, Yisi Sang, Bing He, Minhua Lin, Tianxin Wei, Dakuo Wang, Benoit Dumoulin, Wei Jin, and Hanqing Lu. Adaptive auto-harness: Sustained self-improvement for agentic system deployment on open-ended task streams. *arXiv preprint arXiv:2606.01770*, 2026.

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

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 *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Sungho Park, Wonjoong Kim, Rongyuan Tan, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, et al. Autosaddler: Automatic harness optimization with durable updates from agent execution traces. *arXiv preprint arXiv:2608.23041*, 2026.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

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. *arXiv preprint arXiv:2606.02461*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

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

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026.

Shuhan Xue, Zixin Ding, Yichen Shen, Yinjie Wang, Zhenfei Yin, Yingcheng Wu, Yuxin Chen, Mengdi Wang, and Ling Yang. Past-bench: Benchmarking the foundations of recursive self-improvement in personal agents. *arXiv preprint arXiv:2608.04003*, 2026.

Chenyang Yang, Xinran Zhao, Tongshuang Wu, and Christian Kästner. Better harnesses, smaller models: Building 90% cheaper agents via automated harness adaptation. *arXiv preprint arXiv:2607.08938*, 2026.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, et al. Jit-agent: Scaling harness intelligence via just-in-time harness evolution. *arXiv preprint arXiv:2608.25593*, 2026a.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026b.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026c.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026d.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026e.

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

APPENDIX

Appendix A Continual Improvement Procedure **14**

Appendix B Implementation Contract and Admission Details **14**

B.1 Model Pool and Meta-Model Roles 14

B.2 Round Isolation and Persistence 16

B.3 Typed Execution and Routing Constraints 16

B.4 Evidence Eligibility and Playbook Provenance 17

B.5 Asset Ranking and Candidate Constraints 17

B.6 Paired Validation, Resource Limits, and Lifecycle 18

Appendix C Initial Pools and Representative Execution Cases **19**

C.1 ALFWorld Cases 19

C.2 Omni-MATH Cases 21

C.3 Representative Learned Playbooks 23

Appendix D Evaluation Details **23**

D.1 Per-Benchmark Continual-Improvement Profiles 24

D.2 Completed ALFWorld State and Learning Curves 24

D.3 Asset Growth and Lifecycle Composition 25

D.4 Model Routing and Asset Reuse Diagnostics 26

D.5 Exact Online Token Accounting 26

Appendix E Metric Definitions **28**

Appendix F Validity and Exclusion Rules **29**

Appendix G Baseline, Control, and Ablation Matrix **29**

G.1 Baseline and Ablation Protocols 29

G.2 Adaptation-Surface Control Details 30

G.3 Cost-Oriented Lightweight Variants 32

A CONTINUAL IMPROVEMENT PROCEDURE

Algorithm 1 summarizes the complete HarnessWeaver procedure, including frozen baselines, fast-loop execution, evidence audit, candidate generation, counterfactual validation, and atomic state updates.

Algorithm 1 Continual HarnessWeaver improvement procedure.

Require: task batches $B_1, \ldots, B_J$, initial state H_0, rounds per batch R, fixed development set $\mathcal{D}$
Freeze H_0; evaluate all B_j and $\mathcal{D}$ to obtain $\{S(H_0, B_j)\}_{j=1}^{J}$ and D_0
Preserve frozen assembly specifications, trajectories, and verified outcomes for B_1
for $j = 1, \ldots, J$ **do**
  for $r = 1, \ldots, R$ **do**
    Set $t \leftarrow (j-1)R + r - 1$; snapshot $\bar{H}_t \leftarrow H_t$ before execution
    if $j = 1$ and $r = 1$ **then**
      Reuse the frozen specifications, trajectories, and outcomes of B_1 as round-1 evidence
    else
      Select $(m_x, A_x, \rho_x, p_x) \leftarrow W(x, \phi(x), \bar{H}_t)$ for each $x \in B_j$
      Compile and execute the assemblies in parallel; record trajectories and verifier outcomes
    end if
    Persist versioned trajectories; audit, retry transients, and quarantine engineering faults
    Update actually used assets and write successful exact assemblies as ACTIVE playbooks
    Attribute eligible failures and cluster them by normalized failure mechanism
    Generate and schema-check model-only, harness-only, and joint candidates
    Hash-deduplicate candidates; validate each on all cited sources and matched native peers
    Assign evidence tiers and select the highest-ranked admissible candidate per evidence scope
    Commit selected updates and lifecycle changes to form H_{t+1}
    if $(t+1) \bmod 2 = 0$ **then**
      Record $D_{t+1} \leftarrow S(H_{t+1}, \mathcal{D})$ without updating adaptation state
    end if
  end for
end for

B IMPLEMENTATION CONTRACT AND ADMISSION DETAILS

Table 3: Protocol constants fixed before formal evaluation. Engineering settings such as concurrency, timeouts, retries, ports, and file paths are specified in the released configuration files rather than reproduced here.

Setting	Value
Batch size	20; final ALFWorld batch: 10
Rounds per batch	5
Maximum components per assembly	8
Minimum cluster support	2 (singleton confidence ≥ 0.9)
Maximum component / rule / playbook ops per cluster	6 / 4 / 0
Maximum joint operations per cluster	1
Maximum candidate operations retained per round	12
Validation sources	all cited failures
Validation peers	one native matched peer per source
Promotion	2 successes, win rate ≥ 0.5
Demotion	3 failures and win rate < 0.35
Non-seed asset retirement	not selected for 8 consecutive rounds
Development frequency	every 2 rounds, 20 tasks, read-only

B.1 MODEL POOL AND META-MODEL ROLES

Both benchmarks share the same task-executor pool so that no benchmark receives a hand-strengthened model roster. The table gives the escalation order used in the experiments: it constrains proposal search after failure rather than forcing the first listed model on every task, and the default

executor remains local Qwen3.5-9B. An API routing rule must additionally contain a pre-execution observable predicate that cannot uniquely identify an item, and the Pro model is subject to the restricted-model gate.

Table 4: Shared task-executor pool and the post-failure escalation order used in the experiments.

Model	Access and role	Routing status
Qwen3.5-9B	Local SGLang; default executor	default
Qwen3.8-27B	Local SGLang; stronger executor	escalation 1
Qwen3.7-Flash	DashScope; requests capped at 32K context	escalation 2
Qwen-Long	DashScope; long-context executor	escalation 3
GLM-5.3-Flash	Zhipu; low-cost reasoning executor	escalation 4
Qwen3.8-Flash	DashScope; mid-tier executor	escalation 5
DeepSeek-V4-Flash	DeepSeek; stronger API executor	escalation 6
DeepSeek-V4-Pro	DeepSeek; high-cost fallback	restricted

Table 5 records the role-specific generation settings used in both primary runs. ALFWorld executors disable explicit long-form reasoning to preserve the action protocol, whereas Omni-MATH retains reasoning and uses longer output budgets. Both benchmarks share the same Weaver, attributor, Evolver, JSON-repair, and independent-Reviewer profiles. Sampling fields not explicitly fixed use the provider default and remain recorded in the resolved configuration snapshot.

Table 5: Role-specific generation settings for both primary runs.

Benchmark	Role	Model profile	Mode	Context	Max output
ALFWorld	Default executor	Qwen3.5-9B SGLang	reasoning off	32,768	512
	Strong executor	Qwen3.8-27B SGLang	reasoning off	32,768	512
Omni-MATH	Default executor	Qwen3.5-9B SGLang	reasoning on	32,768	8,192
	Strong executor	Qwen3.8-27B SGLang	reasoning on	32,768	12,288
Both	Weaver	Qwen3.8-27B SGLang-meta	reasoning on	32,768	12,288
	Per-task attributor	GLM-5.3-Flash	low effort	200,000	2,048
	Cluster Evolver	GLM-5.3-Flash	high effort	200,000	8,192
	JSON repair	GLM-5.3-Flash	low effort	200,000	8,192
	Independent Reviewer	Qwen3.8-27B SGLang-meta	reasoning on	32,768	12,288

Task execution uses temperature 0.2, one seed (0), one rollout per task, concurrency 8, and no executor retry; ALFWorld permits at most 30 environment steps, whereas Omni-MATH permits at most four solution steps. Concurrency/retries are 4/2 for per-task attribution and 3/3 for both cluster evolution and JSON repair; the Reviewer is scheduled over the validation scope. The Evolver's 50-record failure limit exceeds the maximum of 20 tasks in a round, so it does not remove failed tasks in the formal runs. Each event excerpt is capped at 3,000 characters and the batch digest at 20,000 characters to bound context and cost and to prevent one verbose observation from crowding out evidence from other failures. These limits apply only to compact post-execution action–observation summaries: complete trajectories remain persisted, while task features, the executed model and components, steps, tokens, exit reason, and verifier outcome are retained in full, and every eligible failure still receives per-task attribution. Truncation therefore cannot alter the executed trajectories, verifier decisions, or reported metrics. Validation is configured for at least 3 and at most 12 scheduled items. All eligible cited sources remain mandatory when a source cluster is larger; the scheduling cap constrains additional validation work and cannot silently discard source evidence cited by a candidate.

Task executors are separated from meta-agent roles. In the primary runs, the Weaver and independent Reviewer use local Qwen3.8-27B, whereas per-task attribution, mechanism-cluster proposal, and schema repair use role-specific GLM-5.3-Flash profiles. Equivalent DeepSeek-V4-Flash role configurations are retained only as preconfigured fallbacks for service unavailability; they are not used in the reported primary runs, and the two configurations are never switched or mixed within a run. GLM-4.7-Flash was not included in the selectable task-executor pool because its public free endpoint was persistently rate-limited during the experiments; Kimi variants and full GLM-5.3 were also excluded from that pool because of cost or experimental-scope constraints. The Challenge Generator

is reserved for environments without native peers. Both formal benchmarks select peers from their own frozen real-data pools and therefore do not invoke a generative challenge model in the primary runs.

B.2 Round isolation and persistence

At the start of execution round $t + 1$, the system creates an immutable snapshot $\bar{H}_t$, which is the only state visible to tasks in that round. Auditing, learning, candidate validation, and commitment follow task execution. Round metrics record the pre-update executions; the updated component library is saved after validation and lifecycle processing.

Before any adaptation, the system evaluates development and every future evolution batch under the same initial state, manifest, batch plan, seed, and verifier. The complete frozen trajectories for Batch 1 are promoted directly to the execution evidence of Round 1, avoiding a stochastic rerun solely to start self-improvement. Frozen trajectories for all later batches remain comparison-only and cannot enter retrieval, KPIs, or the Evolver. Once written, a frozen plan can be resumed only if its dataset manifest, batch size, rounds per batch, and ordered task sequence match exactly.

Recovery is bounded by the last committed round and its recorded state. If complete trajectories exist but the slow loop was interrupted, only unfinished attribution, cluster proposal, or validation is resumed; the environment is not rerun, and successful cluster responses are reused independently. Case, candidate, and playbook identities are deterministic, while model-KPI history uses the round identifier as an idempotency key. A configuration or manifest mismatch fails closed instead of continuing from stale state.

Each run stores its resolved configuration, dataset manifest, and ordered batch plan. Frozen evaluations and online rounds retain task-level terminal outcomes, aggregate metrics, versioned trajectories, and quarantine records; the slow loop additionally records per-task attribution, mechanism clusters, candidate proposals, validation outcomes, and committed state transitions. We compute S_0 only from pre-adaptation frozen evaluations, S_{FE} from the first round of each batch, and S_{final} from the last round of each batch. Runs with missing tasks, unresolved quarantine entries, or inconsistent configurations are excluded from the result tables.

B.3 Typed execution and routing constraints

HarnessWeaver composes only four predefined and auditable component types: prompt blocks, skills, processors, and tool toggles. The Evolver may propose additions or modifications to these structured components, but it cannot generate or execute arbitrary Python code. Components may be invoked at eight fixed stages of task execution: task start, step start, before model invocation, after model invocation, before tool invocation, after tool invocation, step end, and task end. Each processor must explicitly declare its trigger stage, the state fields it may modify, its dependencies and exclusions, and its behavior on failure. Before deployment, the compiler verifies that components are attached to compatible execution stages, required tools are registered, dependencies are satisfied, exclusion and singleton constraints are respected, and injected content remains within the context budget.

When selecting an executor and components for the current task, the Weaver may read the task itself, structured features extracted before execution, type-level summaries of relevant prior tasks, historical performance statistics for assemblies that were actually used, and summaries of the evidence supporting validated assets. To prevent instance memorization and post-outcome leakage, it cannot access current or historical task identifiers, the raw text of previous tasks, complete execution trajectories, reference answers, rewards, or failure labels produced after execution. Persistent routing rules may depend only on general attributes available before execution, such as data source, environment, domain, task family, required tool profile, and dataset-provided difficulty group.

A playbook stores a validated executor–component assembly. It can be invoked only when the specified executor and every listed component remain valid; the system neither silently removes unavailable components nor substitutes another executor. A model-only playbook that specifies a valid executor and no components remains a legal exact match.

The same pre-execution features define the granularity of KPI sharing. New records are aggregated first within a task family of the same benchmark, falling back to domain, tool profile, and benchmark-

level statistics when the family is unavailable. Online records never infer a category from a task identifier. Consequently, success, reward, tokens, latency, monetary cost, and GPU time can inform related tasks without degenerating into one KPI cell per item.

B.4 Evidence eligibility and playbook provenance

The Evidence Auditor classifies every terminal trajectory as success, behavioral failure, protocol failure, or infrastructure failure. Invalid tool names, malformed actions, environment exceptions, and recognized service failures make a record ineligible for evolution. Such records are retained in the quarantine audit layer; they neither lower capability KPIs nor trigger attribution. Only verifier-negative executions with a valid protocol and healthy environment proceed to per-task attribution and mechanism clustering. Successful and quarantined records therefore cannot be mistaken for failures requiring repair.

A deployable playbook has exactly two admission paths. First, an exact (m, C) assembly that succeeds in an ordinary task is deterministically hashed under a reusable pre-execution scope, stored as ACTIVE, and linked to an observational-success certificate. Second, the exact model-only, harness-only, or joint assembly selected by the Validator is stored with its source/peer counterfactual certificate. If a semantically identical playbook already exists, the system augments its real-case links and certificates instead of creating a duplicate. Neither failed trajectories nor Evolver text can create a playbook directly. Thus, `playbook_ops`=0 in Table 3 prohibits free-form, unexecuted playbooks from the meta-model; it does not disable these two evidence-backed paths.

Internally, each case retains its round, pre-execution features, actual model and components, outcome, cost, compact trajectory summary, and evidence links. The Weaver, however, receives only de-identified task types, used assemblies, aggregate KPIs, distilled skill summaries, and opaque case identifiers. Raw task identifiers, historical questions, complete trajectories, and answers remain in the audit layer. This separation preserves a reviewable evidence chain without allowing task identity to become a routing feature.

Formally, a Case Library record and a Playbook Store entry are

$$e_j = (z_j, t_j, \phi_j, m_j, C_j, y_j, r_j, \tau_j, \kappa_j), \tag{13}$$

$$p_k = (\psi_k, m_k, C_k, Z_k, \chi_k, s_k), \qquad Z_k \subseteq \{z_j\}_j, \tag{14}$$

where z_j is an opaque case identifier, t_j remains only in audit storage, ϕ_j contains pre-execution features, (m_j, C_j) is the assembly actually executed, and $y_j, r_j, \tau_j, \kappa_j$ denote outcome, reward, trajectory reference, and resource record. For a playbook, ψ_k is an allow-listed scope predicate, (m_k, C_k) is the exact assembly to execute, Z_k contains links to real cases, χ_k is an observational-success or source/peer counterfactual certificate, and s_k is lifecycle state. The Weaver may consume de-identified summaries of ϕ_j and (m_j, C_j) together with Z_k and χ_k, but not t_j or historical raw tasks. A case may support multiple playbooks, and a playbook must remain traceable to one or more real cases; an unlinked language-model suggestion cannot become deployable.

For atomic snapshots and consistency checks, the implementation may colocate components, rules, and playbooks in one serialized library object. This is a persistence detail and does not collapse their logical roles. The Case Library and model KPIs use separate databases, and development evaluation compares adaptation state before and after execution to verify that read-only monitoring changes no persistent state.

B.5 Asset ranking and candidate constraints

For component a, let n_a and s_a denote counted real-task executions and successes. Let v_a be its validation marginal value, $h_a = \max(0, \Delta\text{tokens}_a)$ its nonnegative marginal token overhead relative to the incumbent, and A_a its lifecycle factor. Ranking uses Beta$(1, 1)$ smoothing:

$$\hat{p}_a = \frac{s_a + 1}{n_a + 2}, \quad e_a = \frac{n_a}{n_a + 4}, \quad w_a = A_a \left[0.5(1 - e_a) + \hat{p}_a e_a\right] \text{clip}(1{+}v_a, 0.5, 1.5)(1{+}h_a/4000)^{-1}. \tag{15}$$

Here $A_a = 1$ for ACTIVE, 1.1 for PROBATION with fewer than two real uses and 0.7 thereafter, and 0 for SHADOW or RETIRED. This score ranks admitted assets only and does not replace counterfactual validation.

Each candidate must cite all source evidence within its declared scope and satisfy the component schema, selectable-model constraints, operation limits, and conflict rules. A candidate may address either an entire mechanism cluster or a coherent subset defined by explicit applicability conditions. For each cluster, the Evolver may propose at most six component operations, four model-routing operations, and one joint operation; after merging, normalization, and deduplication, at most twelve candidate operations are retained per round. The Evolver cannot directly create or modify playbooks. Instead, a playbook is induced only from an assembly that succeeds in actual execution or is selected through validation, preventing unsupported assemblies from entering persistent state. Approximately valid JSON may undergo only field-level formatting repairs whose values are recoverable from the existing response; missing semantics are never invented. Structurally valid candidates proceed to independent validation after deduplication.

B.6 Paired validation, resource limits, and lifecycle

The Validator does not rerun the original harness; controls are the immutable same-round trajectories that triggered attribution. For source set $\mathcal{S}$, whose members are failure evidence, recovery and mean reward improvement are

$$\mathrm{Rec}(c) = \sum_{i \in \mathcal{S}} \mathbb{I}[r_i^0 < \theta \wedge r_i^c \geq \theta], \tag{16}$$

$$\Delta_S(c) = |\mathcal{S}|^{-1} \sum_{i \in \mathcal{S}} (r_i^c - r_i^0). \tag{17}$$

Strict initial admission of a failure-derived candidate requires $\mathrm{Rec}(c) = |\mathcal{S}|$, success on every peer, and satisfaction of the safety constraints. PROBATION requires at least one recovered source, $\Delta_S(c) > 0$, and the safety constraints. A probationary candidate is labeled `supported` if at least one peer succeeds and `exploratory` if peer evidence exists but all peers fail. Both may be selected for controlled trials, but supported evidence wins within an otherwise matched candidate comparison; neither satisfies the evidence required for ACTIVE admission. Only in the `w/o Native Peer` ablation, positive source-only recovery is recorded separately as `source_only` and cannot qualify for ACTIVE. For model and joint candidates, token, latency, and GPU-time ratios are capped at 2.0, 3.0, and 3.0, with additional preregistered caps on per-task and incremental API cost. Harness-only candidates use token and latency caps of 1.25 and 1.5.

Passing candidates with the same source-evidence scope are ordered lexicographically by admission tier, probation evidence level, source success, peer success, overall success, source improvement, monetary cost, GPU time, latency, and tokens; a single-surface change wins an exact tie. PROBATION is promoted after at least two subsequent real successes with win rate at least 0.5. Thus, supported versus exploratory evidence affects initial competition, while promotion ultimately depends on later deployment evidence. A non-seed ACTIVE asset is demoted after at least three failures with win rate below 0.35, and a non-seed deployable asset retires if never selected during eight rounds after creation. Online KPIs are assigned only to assets actually used, assembly credit is maintained separately for playbooks, and development evaluation is always read-only.

The resource constraints are fixed hard thresholds rather than learned weights in the conceptual utility. Let T_c, L_c, G_c, M_c be candidate tokens, latency, GPU seconds, and API charge per attempt, and let the superscript 0 denote the immutable incumbent observation. For $X \in \{T, L, G\}$, the implementation uses the following zero-safe ratio:

$$\rho_X(c) = \begin{cases} X_c / X_0, & X_0 > 0, \\ 1, & X_0 \leq 0 \ \wedge \ X_c \leq 0, \\ +\infty, & X_0 \leq 0 \ \wedge \ X_c > 0, \end{cases} \qquad \Delta M(c) = M_c - M_0. \tag{18}$$

Model-only and joint candidates require $\rho_T \leq 2$, $\rho_L \leq 3$, and $\rho_G \leq 3$, while both M_c and $\Delta M(c)$ must remain within preregistered API-cost caps. Harness-only candidates require $\rho_T \leq 1.25$ and $\rho_L \leq 1.5$. Resource terms break ties only among candidates with identical source scope and comparable evidence tier and success; they cannot compensate for a weaker admission tier.

Let $n_t(a)$, $q_t(a)$, $w_t(a)$, and $f_t(a)$ denote KPI-counted real-task executions, actual selections, wins, and failures of asset a after t completed rounds. The principal lifecycle transitions are

$$\textsc{Probation} \rightarrow \textsc{Active} \quad \text{if } w_t(a) \geq 2 \ \wedge \ w_t(a)/n_t(a) \geq 0.5, \tag{19}$$

$$\textsc{Active} \rightarrow \textsc{Probation} \quad \text{if } f_t(a) \geq 3 \ \wedge \ w_t(a)/n_t(a) < 0.35, \tag{20}$$

$$\{\textsc{Active}, \textsc{Probation}\} \rightarrow \textsc{Retired} \quad \text{if } q_t(a) = 0 \text{ eight rounds after creation}, \tag{21}$$

where seed components are exempt from automatic retirement. Explicit retirement may also follow a validated retirement proposal, unavailable model dependency, or invalidated persistent dependency, with every transition written to the append-only journal. SHADOW assets are not deployable by the Weaver and remain only for audit or subsequent evidence updates.

C Initial Pools and Representative Execution Cases

Both benchmarks begin with a small set of generic, manually specified components that encode task-level execution principles rather than benchmark instances or solutions. Seed components are ACTIVE, whereas routing rules, playbooks, cases, and online KPIs are initially empty; subsequent routes and exact assemblies must therefore arise from execution and validation evidence. No seed component contains task identifiers, reference answers, or instance-specific solution traces. ALFWorld starts with nine general interaction components and Omni-MATH with four general mathematical-reasoning components. Table 6 lists the complete pools.

Table 6: Complete initial component pools. PB, Skill, and Proc denote prompt block, skill, and processor.

Benchmark	Type	Component ID	Purpose
ALFWorld	PB	`pb-plan-first`	Form and silently revise a short subgoal plan.
	PB	`pb-verify-before-finish`	Check every goal condition before termination.
	PB	`pb-observe-first`	Prefer grounded observations before state changes.
	Skill	`skill-systematic-recovery`	Repair a failed action by identifying and changing its unmet precondition.
	Proc	`proc-obs-truncator`	Bound oversized observations while retaining their recent tail.
	Proc	`proc-progress-reminder`	Inject a progress check every four steps.
	Proc	`proc-repeat-guard`	Detect identical actions repeated in an unchanged state.
	Proc	`proc-loop-detection`	Interrupt longer repetition loops.
	Proc	`proc-tool-failure-guard`	Contain cascades following tool failures.
Omni-MATH	PB	`pb-math-decompose`	Identify givens, target, constraints, and a justified solution route.
	PB	`pb-math-representation-check`	Preserve object types, quantifiers, domains, and logical direction.
	PB	`pb-math-verify-boxed`	Verify decisive steps and emit one unambiguous boxed answer.
	Skill	`skill-math-repair`	Locate the earliest invalid derivation step and recompute downstream results.

The following cases are drawn from the completed full-method runs rather than constructed post hoc. For clarity, we summarize the task setting, executed assembly, key execution behavior, and verifier outcome.

C.1 ALFWorld cases

Success. In one task, the agent must take the pepper shaker by the coffee machine and place it in the second drawer. Qwen3.5-9B executes with `pb-plan-first`, `pb-observe-first`, `pb-verify-before-finish`, and `proc-repeat-guard`. It completes the five-action chain below; the environment verifier returns success, with no repeated action or environment error.

TASK : place the pepper shaker by the coffee machine in drawer 2
⇓
WEAVER : Qwen3.8-27B SGLang-meta selects the task-conditioned assembly
⇓
EXECUTOR : Qwen3.5-9B
⇓
HARNESS : {`plan-first`, `observe-first`, `verify-before-finish`, `repeat-guard`}
⇓
LOCATE : `go to coffeemachine 1`
⇓
ACQUIRE : `take peppershaker 1 from countertop 1`
⇓
NAVIGATE : `go to drawer 2`
⇓
PREPARE : `open drawer 2`
⇓
COMPLETE : `move peppershaker 1 to drawer 2`
⇓
VERIFIER : SUCCESS.

Failure. The failed task asks the agent to take the salt shaker beside the coffee machine and place it in the second drawer. Qwen3.5-9B uses only `pb-observe-first` and `pb-verify-before-finish`, but places the object in the first drawer. It then repeatedly examines, opens, and closes that incorrect drawer, producing 21 repeated actions within the 30-step limit (rate 0.70) and reward zero. Because the protocol and environment remain valid, the Evidence Auditor treats this as an evolution-eligible behavioral failure.

TASK : place the salt shaker by the coffee machine in drawer 2
⇓
WEAVER : Qwen3.8-27B SGLang-meta selects the task-conditioned assembly
⇓
EXECUTOR : Qwen3.5-9B
⇓
HARNESS : {`observe-first`, `verify-before-finish`}
⇓
SEARCH : open drawer 1 and inspect the countertop and shelves
⇓
WRONG PLACEMENT : put the salt shaker in drawer 1
⇓
LOOP : (`examine` → `open` → `close drawer 1`) × 7
⇓
OUTCOME : 21 repeated actions in 30 steps, $r = 0$.

Recovery. The system first attributes the wrong-goal choice and subsequent repetitive recovery behavior to a common failure mechanism, then compares candidate interventions over the harness-only, model-only, and joint adaptation surfaces. Validation shows that the existing component assembly remains useful: switching only the executor to Qwen3.8-27B solves both the source task and a matched native peer. This model-only candidate uses 24.5% of the incumbent tokens and 21.8%

of its latency, passes the resource Gate, and is selected as the highest-ranked admissible intervention. The resulting route and exact reusable playbook are committed as ACTIVE assets.

$$
\begin{aligned}
\textsc{Evidence audit}: &\ \text{valid environment} + \text{valid protocol} + \text{behavioral failure}\\
&\Downarrow\\
\textsc{Attribution}: &\ \text{GLM-5.3-Flash groups wrong-goal choice and repetitive recovery}\\
&\Downarrow\\
\textsc{Harness candidate}: &\ \text{revise execution control}\\
\textsc{Model candidate}: &\ \text{change the executor}\\
\textsc{Joint candidate}: &\ \text{change both surfaces}\\
&\Downarrow\\
\textsc{Evolver}: &\ \text{GLM-5.3-Flash proposes evidence-linked interventions}\\
&\Downarrow\\
\textsc{Validation}: &\ \text{source success} + \text{matched native-peer success}\\
&\Downarrow\\
\textsc{Reviewer}: &\ \text{Qwen3.8-27B SGLang-meta independently reviews the evidence}\\
&\Downarrow\\
\textsc{Selection}: &\ \text{model-only is the highest-ranked admissible intervention}\\
&\Downarrow\\
\textsc{Selected assembly}: &\ \text{fixed harness} + \text{Qwen3.8-27B executor}\\
&\Downarrow\\
\textsc{Resource check}: &\ 0.245\times \text{ incumbent tokens}, \quad 0.218\times \text{ incumbent latency}\\
&\Downarrow\\
\textsc{Persistent update}: &\ \textsc{Active route} + \text{reusable exact playbook.}
\end{aligned}
$$

C.2 OMNI-MATH CASES

Success. The numbers $1, 2, \ldots, 11$ are arranged uniformly at random from left to right. Conditional on the middle number being larger than exactly one number to its left, the task asks for the probability that it is also larger than exactly one number to its right. In the second round, the Weaver retains all four mathematical seed components and routes the task to Qwen3.7-Flash. The resulting execution gives a complete solution in one solution step: $\binom{11}{6} = 462$ permutations satisfy the left-side condition and $2\binom{8}{4} = 140$ satisfy both conditions, yielding $140/462 = 10/33$. The pinned Omni-Judge accepts the answer.

$$
\begin{aligned}
\textsc{Task}: &\ \text{compute the stated conditional probability}\\
&\Downarrow\\
\textsc{Weaver}: &\ \text{Qwen3.8-27B SGLang-meta selects the task-conditioned assembly}\\
&\Downarrow\\
\textsc{Executor}: &\ \text{Qwen3.7-Flash}\\
&\Downarrow\\
\textsc{Harness}: &\ \text{all four mathematical seed components}\\
&\Downarrow\\
\textsc{Condition count}: &\ N_{\text{left}} = \binom{11}{6} = 462\\
&\Downarrow\\
\textsc{Joint count}: &\ N_{\text{left,right}} = 2\binom{8}{4} = 140
\end{aligned}
$$

$$\Downarrow$$

$$\text{ANSWER}: \ \frac{140}{462} = \boxed{\frac{10}{33}}$$

$$\Downarrow$$

$$\text{VERIFIER}: \ \text{OMNI-JUDGE SUCCESS}.$$

Failure. In the first round, the same task is executed by local Qwen3.5-9B with the same four mathematical seed components. Four generation attempts reach the output limit without producing a gradable final answer, yielding `reasoning_budget_exhausted` and reward zero. Because the protocol and verifier remain valid, the Evidence Auditor treats this as an evolution-eligible behavioral failure.

$$\begin{aligned}
\text{TASK}: &\ \text{compute the same conditional probability} \\
&\Downarrow \\
\text{WEAVER}: &\ \text{Qwen3.8-27B SGLang-meta selects the task-conditioned assembly} \\
&\Downarrow \\
\text{EXECUTOR}: &\ \text{Qwen3.5-9B} \\
&\Downarrow \\
\text{HARNESS}: &\ \text{all four mathematical seed components} \\
&\Downarrow \\
\text{GENERATION}: &\ \text{attempts } 1, 2, 3, 4 \text{ each reach the output limit} \\
&\Downarrow \\
\text{OUTCOME}: &\ \text{no gradable final answer}, \quad r = 0.
\end{aligned}$$

Recovery. The system attributes the absence of a gradable answer to an executor-capability or generation-budget bottleneck, then compares candidate interventions over the harness-only, model-only, and joint adaptation surfaces. Validation shows that the four-component mathematical harness remains useful and that switching only the executor to Qwen3.7-Flash recovers the source task. The model-only candidate is therefore selected as the highest-ranked admissible intervention. This paired case shows that HarnessWeaver can preserve useful harness support while changing only the adaptation surface identified as the bottleneck.

$$\begin{aligned}
\text{EVIDENCE AUDIT}: &\ \text{valid protocol} + \text{valid verifier} + \text{behavioral failure} \\
&\Downarrow \\
\text{ATTRIBUTION}: &\ \text{GLM-5.3-Flash identifies an answer-completion bottleneck} \\
&\Downarrow \\
\text{HARNESS CANDIDATE}: &\ \text{revise mathematical support} \\
\text{MODEL CANDIDATE}: &\ \text{change the executor} \\
\text{JOINT CANDIDATE}: &\ \text{change both surfaces} \\
&\Downarrow \\
\text{EVOLVER}: &\ \text{GLM-5.3-Flash proposes evidence-linked interventions} \\
&\Downarrow \\
\text{VALIDATION}: &\ \text{the four-component harness remains useful} \\
&\Downarrow \\
\text{REVIEWER}: &\ \text{Qwen3.8-27B SGLang-meta independently reviews the evidence} \\
&\Downarrow \\
\text{SELECTION}: &\ \text{model-only is the highest-ranked admissible intervention} \\
&\Downarrow \\
\text{SELECTED ASSEMBLY}: &\ \text{fixed four-component harness} + \text{Qwen3.7-Flash executor}.
\end{aligned}$$

C.3 Representative Learned Playbooks

A playbook stores a scoped executor–component assembly rather than a task-specific answer. Table 7 reports two frequently reused examples that remain ACTIVE in the final pools; reuse statistics are computed over the complete online runs.

Table 7: Representative ACTIVE playbooks and their subsequent online reuse.

ALFWorld playbook	
Scope	`look_at_obj_in_light`.
Executor	Qwen3.5-9B.
Saved components	Observe-first, plan-first, and verify-before-finish.
Origin	Created from a successful three-step assembly in Round 1.
Evidence base	The supporting evidence later spans eight distinct successful tasks from the same interaction family.
Online reuse	Selected 30 times and successful on 24 selections, for an online success rate of 80.0%.
Final status	Remains ACTIVE through the final ALFWorld round.
Omni-MATH playbook	
Scope	Discrete mathematics / combinatorics.
Executor	Qwen3.7-Flash.
Saved components	Mathematical decomposition, representation checking, final-answer verification, and mathematical repair.
Origin	Created from a one-step judged success in Round 2.
Evidence base	The supporting evidence later spans 31 distinct successful tasks from the same mathematical family.
Online reuse	Selected 140 times and successful on 109 selections, for an online success rate of 77.9%.
Final status	Remains ACTIVE through Round 25.

These examples illustrate continual accumulation at the playbook layer: a successful assembly is compressed into a scoped executable asset and repeatedly invoked on later tasks from the same family. The ALFWorld playbook reuses general interaction controls with the local default model, whereas the Omni-MATH playbook couples the mathematical support components with an evidence-selected executor. Their sustained reuse and final ACTIVE status show that accumulated experience enters subsequent execution rather than remaining only as an offline record.

D Evaluation Details

Each benchmark uses an independent run root, Living Pool, Routing Rule Store, Playbook Store, Case Library, KPI store, and journal. Tasks are first filtered to the designated split while preserving their relative manifest order, then partitioned into contiguous batches. They are not reordered by task family, difficulty, or task identifier. Frozen baselines and online rounds read the same persisted batch plan.

For ALFWorld, we preserve the relative order of all 134 official `valid_unseen` tasks and allocate 70 to evolution, 20 to read-only development, and 44 to the Validator-only native-peer bank; success is determined by environment goal completion. For Omni-MATH, the 100 official test problems form five evolution batches; 20 fixed training problems form development and the remaining 458 form the peer bank. Formal grading uses a fixed revision of the `KbsdJames/Omni-Judge` checkpoint released with Omni-MATH (Gao et al., 2025).

The default executor is local Qwen3.5-9B and the stronger local option is Qwen3.8-27B. The API pool contains Flash models ordered by cost and availability, while high-cost Pro models are restricted candidates. ALFWorld disables long-form reasoning, whereas mathematical tasks retain it. The primary configuration uses GLM-5.3-Flash for per-task attribution and mechanism-cluster proposal; the corresponding DeepSeek-V4-Flash role configuration is retained only as a reproducible fallback for service unavailability and is not used in the reported primary runs.

D.1 Per-Benchmark Continual-Improvement Profiles

Main-text Figure 4 and Appendix Figure 7 separate cross-batch transfer from within-batch recovery for ALFWorld and Omni-MATH, respectively. Frozen H_0 provides the pre-adaptation reference, First encounter measures transfer of persistent updates to a newly encountered batch and corresponds to the values in Table 2, and Final round measures recovery after five executions of that batch.

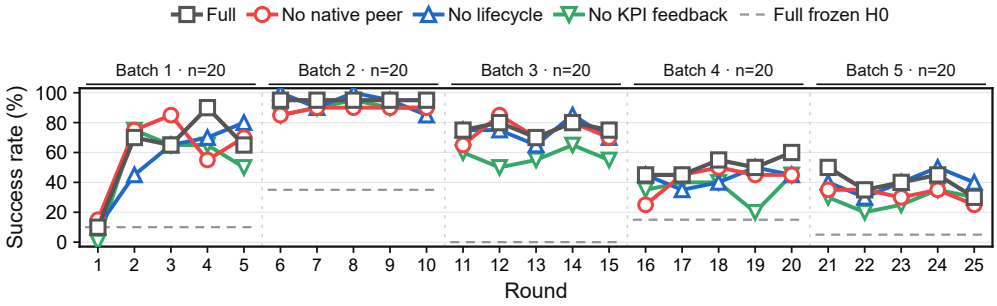

Figure 7: Round-wise Omni-MATH performance for full HarnessWeaver and the three lightweight variants. Gray dashed segments denote the full method's frozen-H_0 reference. The first point in each batch is its first encounter; later points measure within-batch adaptation rather than independent test samples.

Table 8 reports all 25 persisted full-method points underlying Figure 7. Each batch contains 20 tasks, so the recorded success rates change in five-point increments.

Table 8: Round-wise Omni-MATH success (%). R1 is the first encounter and R5 is the final round.

Batch	R1	R2	R3	R4	R5
B_1	10	70	65	90	65
B_2	95	95	95	95	95
B_3	75	80	70	80	75
B_4	45	45	55	50	60
B_5	50	35	40	45	30

Figure 7 and Table 8 show that Full begins from a 13.0% average frozen reference across the five batches, yet reaches 55.0% at first encounter and 65.0% on the final rounds. At first encounter, Batches 2–5 already exceed their respective frozen references by 60, 75, 30, and 45 points, indicating that previously committed routes and harness assets transfer to batches not used to produce those updates. Further within-batch recovery is concentrated in Batch 1 (10% to 65%) and Batch 4 (45% to 60%), while Batches 2 and 3 retain their strong first-encounter performance. The curves therefore expose both cross-batch transfer and additional recovery on harder batches. Full also attains the strongest aggregate first-encounter/final-round pair among the four plotted configurations (55.0%/65.0%), compared with 45.0%/60.0% without native peers, 54.0%/64.0% without lifecycle management, and 42.0%/54.0% without KPI feedback. The largest degradation follows removal of KPI feedback, supporting the value of observed online outcomes for subsequent routing and component selection.

D.2 Completed ALFWorld State and Learning Curves

Table 9 provides every persisted full-method point underlying Figure 4, rather than only its endpoints. The four batches contain 20, 20, 20, and 10 tasks. Aggregates across batches are therefore task-weighted. The curves are not monotonic, motivating our separate reporting of first-encounter transfer, final recovery, and best checkpoint performance.

Read-only development is 70% before adaptation and is reevaluated periodically without writing back to any adaptation state. Because these checkpoints repeatedly evaluate the same 20 tasks, they are monitoring measurements rather than unseen-test results. The online stream contains 350 task

Table 9: Round-wise ALFWorld success (%). R1 is the first encounter and R5 is the final round.

Batch	R1	R2	R3	R4	R5
B_1	95	85	85	85	85
B_2	45	80	80	75	75
B_3	55	65	80	85	90
B_4	20	70	60	40	70

executions, of which 259 succeed. This 74.0% descriptive micro-average over repeated rounds differs from both the 58.6% first-encounter metric and the 81.4% final-round metric.

D.3 Asset Growth and Lifecycle Composition

Figure 8 traces persistent-state growth after each round, while Table 10 reports the final lifecycle composition and observed online reuse on both benchmarks. Reused counts assets selected by at least one real evolution task in a round after their creation; Uses is the corresponding number of subsequent selections.

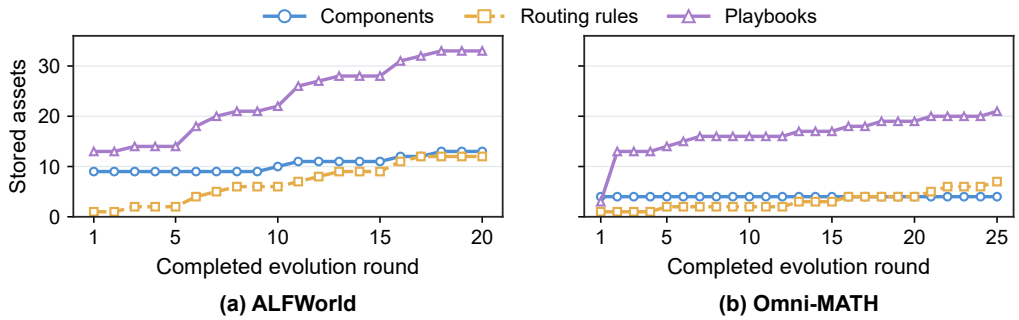

Figure 8: Persistent components, routing rules, and playbooks over completed evolution rounds. Counts come from the committed state snapshot after each round.

Table 10: Final persistent assets and subsequent online reuse. A, P, and R denote ACTIVE, PROBATION, and RETIRED.

Benchmark	Object	Initial	Final	A	P	R	Reused	Uses
ALFWorld	Components	9	13	10	1	2	9	1,924
	Routing rules	0	12	0	5	7	1	1
	Playbooks	0	33	11	1	21	9	295
Omni-MATH	Components	4	4	4	0	0	4	1,935
	Routing rules	0	7	1	4	2	3	24
	Playbooks	0	21	14	3	4	15	456

The final snapshots additionally retain 47 ALFWorld and 24 Omni-MATH candidate records. These records preserve proposals, validation evidence, and Gate decisions but are not deployable assets, so they are excluded from the Final and Reused columns of Table 10. ALFWorld adds four components, 12 routing rules, and 33 playbooks, whereas Omni-MATH's persistent growth is concentrated in seven routing rules and 21 playbooks.

The two benchmarks exhibit distinct but complementary accumulation patterns. On ALFWorld, the component pool grows only from nine to 13, yet components receive 1,924 subsequent selections, indicating extensive reuse of general interaction controls together with a small number of added mechanisms. Nine of the 33 learned playbooks are reused for 295 later selections, while 21 playbooks and seven routing rules are ultimately retired; lifecycle management therefore removes narrow or weakly supported assets rather than allowing unchecked pool growth. Omni-MATH retains and reuses all four seed reasoning components, with adaptation occurring primarily at the task-conditioned routing and playbook layers: 15 playbooks are reused 456 times and 14 remain ACTIVE. This

pattern is consistent with mathematical tasks sharing stable reasoning primitives while benefiting from problem-conditioned executor and assembly selection. Across both benchmarks, the persistent pool is thus not a record of every proposal: transferable assemblies are repeatedly deployed, while PROBATION and RETIRED states constrain ineffective persistence.

D.4 Model Routing and Asset Reuse Diagnostics

Figure 9 shows the routing distribution when each batch is first encountered, and Table 11 aggregates executor selection and verified outcomes over all online rounds. Because the Weaver adaptively assigns different tasks to different executors, the rates describe realized routing rather than a randomized comparison of model capability. Table 10 additionally shows that persistent assets enter later execution: playbooks account for 295 subsequent selections on ALFWorld and 456 on Omni-MATH.

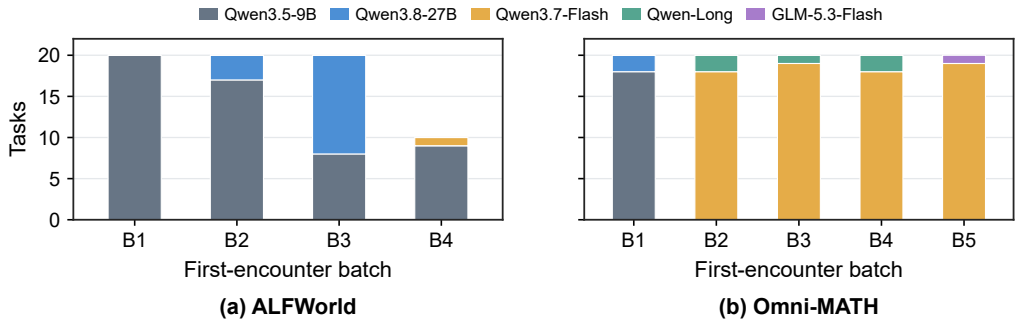

Figure 9: Executor choices at the first encounter of each batch in the full runs. Each stacked bar reports the realized routing distribution for newly encountered tasks.

Table 11: Online executor routing in the full runs. Success rates are conditional on adaptive assignment to each executor.

Benchmark	Executor	Executions	Successes	Success (%)
ALFWorld	Qwen3.5-9B	94	57	60.6
	Qwen3.8-27B	226	178	78.8
	Qwen3.7-Flash	30	24	80.0
Omni-MATH	Qwen3.5-9B	21	2	9.5
	Qwen3.8-27B	2	0	0.0
	Qwen3.7-Flash	446	300	67.3
	Qwen-Long	18	9	50.0
	GLM-5.3-Flash	13	11	84.6

The routing trajectories show that the system does not retain a single executor throughout the stream. ALFWorld begins with all 20 first-batch tasks assigned to the default Qwen3.5-9B; as online evidence accumulates, Qwen3.8-27B takes the largest share of subsequent execution and succeeds on 178 of 226 assignments (78.8%), while Qwen3.7-Flash is selected for a smaller task subset and succeeds on 24 of 30 assignments. The shift is sharper on Omni-MATH: the first batch is still dominated by local Qwen3.5-9B, but first encounters from Batch 2 onward are routed almost entirely to Qwen3.7-Flash, which ultimately handles 446 of 500 online executions and succeeds on 300. Qwen-Long and GLM-5.3-Flash remain selectively invoked, indicating that alternative executors are reserved for evidence-supported cases rather than used through uniform allocation. Together with the repeated playbook selections in Table 10, these results show that accumulated execution evidence changes both executor choice and reusable assembly selection, yielding benchmark-specific stable execution strategies.

D.5 Exact Online Token Accounting

Table 12 gives the exact counters underlying Figure 6, separating task execution, evolution control, and candidate-validation replay. Evolution contains meta-agent work for attribution, proposal generation,

review, and state management; Validation replay counts candidate executions on source tasks and native peers. These online counters exclude frozen-baseline and read-only development evaluations. Saving is relative to Full on the same benchmark; a negative value denotes additional total tokens.

Table 12: Exact online token use for Full and the four cost-oriented variants, with evolution and validation replay reported separately.

ALFWorld					
Token category	Full	w/o Native Peer	w/o Lifecycle	w/o KPI Feedback	Accept-All
Task execution	18,341,048	18,652,460	18,069,995	17,904,188	21,200,779
Evolution	1,374,021	1,046,825	1,050,465	1,145,860	765,628
Validation replay	13,591,130	8,803,764	10,704,633	12,036,207	0
Total	33,306,199	28,503,049	29,825,093	31,086,255	21,966,407
Saving vs. Full (%)	0.0	14.4	10.5	6.7	34.0
Omni-MATH					
Token category	Full	w/o Native Peer	w/o Lifecycle	w/o KPI Feedback	Accept-All
Task execution	12,922,673	13,155,421	12,247,184	8,029,350	13,155,382
Evolution	1,204,207	1,522,212	1,749,358	1,729,940	1,624,764
Validation replay	1,625,006	2,349,269	2,770,790	4,384,708	0
Total	15,751,886	17,026,902	16,767,332	14,143,998	14,780,146
Saving vs. Full (%)	0.0	−8.1	−6.4	10.2	6.2

Table 13 completes the accounting for the remaining methods reported in the main comparison: the frozen reference, the two external baselines, and the completed internal controls. Frozen H_0 executes each evolution task once; DC and ACE follow their respective single-pass online protocols; Harness-only and Model-only follow the same five-round ALFWorld protocol as Full. Method update denotes each external baseline's experience-update cost and, for the internal controls, evolution control excluding Validator replay; Validation replay is reported separately. Dashes mark unavailable Omni-MATH controls for which no complete, protocol-valid run exists.

Table 13: Exact token use for the frozen reference, external baselines, and completed adaptation-surface controls, with method updates and validation replay separated.

Token category	Frozen H_0	DC	ACE	Harness-only	Model-only
ALFWorld					
Protocol	one pass	one pass	one pass	five rounds/batch	five rounds/batch
Task execution	4,783,707	5,805,646	7,477,981	25,774,354	17,877,836
Method update	0	155,856	627,915	859,602	740,488
Validation replay	0	0	0	9,594,292	5,969,368
Total	4,783,707	5,961,502	8,105,896	36,228,248	24,587,692
Omni-MATH					
Protocol	one pass	one pass	one pass	–	–
Task execution	3,553,990	3,351,481	4,014,321	–	–
Method update	0	710,794	961,585	–	–
Validation replay	0	0	0	–	–
Total	3,553,990	4,062,275	4,975,906	–	–

The decomposition reveals how Full allocates its online budget. On ALFWorld, task execution accounts for 55.1% of total tokens, candidate-validation replay for 40.8%, and evolution control for only 4.1%; on Omni-MATH, the corresponding shares are 82.0%, 10.3%, and 7.6%. Meta-agent reasoning itself is therefore not the dominant overhead; most additional expenditure buys executable evidence about candidate updates. All four lightweight ALFWorld variants reduce total tokens.

Accept-All removes validation replay and lowers the total by 34.0%, but its task-execution tokens increase by 15.6% over Full, showing that the saving comes from foregoing admission evidence rather than from shorter downstream execution. On Omni-MATH, removing native-peer validation or lifecycle management increases total use by 8.1% and 6.4%, respectively, primarily through additional replay, indicating that paired evidence and asset retirement can prevent unproductive candidates from continuing to consume resources. w/o KPI Feedback and Accept-All still reduce total use by 10.2% and 6.2% while retaining the continual-adaptation gains reported in the main text, providing lighter operating points.

The smaller totals of the one-pass methods in Table 13 primarily reflect different exposure: Frozen, DC, and ACE execute each task once, whereas the continual methods execute five rounds per batch. Their raw totals therefore do not constitute an efficiency ranking under a matched protocol. Among the protocol-matched ALFWorld controls, Full uses 8.1% fewer total tokens than Harness-only while achieving higher first-encounter and final-round success in the main results. Model-only uses fewer tokens, but also remains below Full at both endpoints; Full spends the additional validation budget to select joint model–harness updates and obtains stronger transfer and recovery. The two tables thus support complementary operating choices: lightweight variants reduce online expenditure, while Full uses controlled validation cost to obtain more reliable joint self-improvement.

E Metric Definitions

Let $n_j = |B_j|$. Weighted frozen, first-encounter, and final-round scores are

$$S_0 = \frac{\sum_j n_j S(H_0, B_j)}{\sum_j n_j}, \tag{22}$$

$$S_{\mathrm{FE}} = \frac{\sum_j n_j S(H_{(j-1)R}, B_j)}{\sum_j n_j}, \tag{23}$$

$$S_{\mathrm{final}} = \frac{\sum_j n_j S(H_{jR-1}, B_j)}{\sum_j n_j}. \tag{24}$$

Aggregate first-encounter transfer is $S_{\mathrm{FE}} - S_0$, and aggregate within-batch recovery is $S_{\mathrm{final}} - S_{\mathrm{FE}}$. Because batch sizes may differ, all aggregate results in the paper are weighted by task count rather than macro-averaged across batches.

Both ALFWorld and Omni-MATH produce binary success outcomes through their fixed verifiers. For any task set D, we report the micro-average of these task-level outcomes:

$$S(H, D) = \frac{1}{|D|} \sum_{x_i \in D} \mathbb{I}[V(x_i, \tau_i) = \mathrm{success}]. \tag{25}$$

The cross-benchmark macro average weights each completed benchmark equally, rather than allowing a larger dataset to dominate. For K_b complete, protocol-valid benchmarks,

$$S_{\mathrm{macro}} = \frac{1}{K_b} \sum_{k=1}^{K_b} S_k. \tag{26}$$

Pending or invalid runs are excluded from both numerator and denominator and remain visibly marked as missing; they are never imputed as zero.

Resource records are not collapsed into a single arbitrarily weighted scalar. The system stores separate token, API-charge, GPU-second, and wall-clock fields for task and candidate executions; our cross-configuration cost comparison focuses on online tokens, which can be reconstructed consistently for every completed run. For run $\mathcal{U}$, the recorded resource vector is

$$\mathbf{C}(\mathcal{U}) = \left(\sum_{i \in \mathcal{U}} T_i^{\mathrm{task}}, \sum_{i \in \mathcal{U}} T_i^{\mathrm{meta}}, \sum_{i \in \mathcal{U}} M_i, \sum_{i \in \mathcal{U}} G_i, \sum_{i \in \mathcal{U}} L_i \right). \tag{27}$$

For the completed ALFWorld run, the persisted counters contain 18,341,048 task tokens and 14,965,151 evolution-and-validation tokens, totaling 33,306,199. The meta counter includes validation replays, not only meta-agent text generation; these online counters exclude frozen and

development evaluations. Local-model API charge is zero by construction, but this does not imply zero compute cost; latency and GPU accounting remain separate fields.

Harness complexity is described by selected-component count, injected prompt tokens, deployable-asset count, and routing-rule count. Consistent with Table 10, let $u^+(a)$ denote real selections of asset a in online rounds strictly after its creation. Subsequent reuse among final persistent assets is

$$\mathrm{Reuse}^+(\mathcal{A}) = \frac{\sum_{a\in\mathcal{A}} \mathbb{I}[u^+(a) > 0]}{|\mathcal{A}|}. \tag{28}$$

The table also reports $\sum_{a\in\mathcal{A}} u^+(a)$ as subsequent Uses. Selection-conditional success s_a/u_a is reported only as a descriptive KPI. It is not interpreted as the causal contribution of asset a, because task assignment and asset choice are adaptive rather than randomized.

F Validity and Exclusion Rules

A run contributes to a formal result only when every scheduled task has a terminal record, the fixed environment verifier is available, the task manifest matches the frozen batch plan, and no infrastructure or protocol quarantine remains unresolved. Infrastructure and protocol faults are not scored as task failures and do not update task history, asset KPIs, or model KPIs; they are quarantined pending rerun. Only protocol-valid behavioral failures are eligible for attribution, mechanism clustering, and candidate evolution, preventing engineering faults from contaminating the self-improvement signal.

The frozen reference is executed once on every future evolution batch before any adaptation occurs; its task plan, trajectories, and metrics then remain immutable. Resumption refuses a manifest or canonical batch-plan mismatch and retains the configuration digest and original fault records. First-encounter results are therefore always compared with the same pre-adaptation H_0.

Development is read-only: it cannot update cases, playbooks, routing evidence, component KPIs, or model KPIs. Native peers are excluded from candidate generation, used only for candidate validation, and never become subsequent evolution tasks. The paper's continual-improvement conclusions therefore come from prequential first-encounter evaluation against the frozen reference and subsequent rounds on the same batch, rather than from development monitoring curves.

G Baseline, Control, and Ablation Matrix

G.1 Baselines

External baselines are Dynamic Cheatsheet and ACE with the environment, tool protocol, and default task executor matched to our study. Each begins from an independently initialized, method-specific online state rather than reading HarnessWeaver's seed-component pool. The reported internal results include Frozen, Harness-only, Model-only, and full HarnessWeaver. Frozen reuses the immutable per-batch baselines collected before adaptation, Harness-only and Model-only are independent ALFWorld runs, and Full is the primary experiment. Four lightweight variants remove native peers, lifecycle management, or KPI feedback, or replace the counterfactual Gate with Accept-All.

To ensure that comparisons reflect adaptation mechanisms rather than infrastructure differences, every baseline reads the same manifest and ordered task stream and shares the default task executor, base task input, environment adapter, tool schema, maximum interaction steps, seed, and official verifier. Method-specific memory enters the execution context only through the baseline's defined injection path. Runs write to isolated roots and cannot read HarnessWeaver's evolved Living Pool, playbooks, KPIs, or trajectories; each method may retain only the online state required by its own definition.

External-baseline results come from protocol-matched reimplementations rather than values copied from the source papers; each reimplementation preserves the method's defining state-update mechanism. Frozen H_0 contains only round-zero executions whose manifest, batch plan, seed, initial component pool, verifier, and registered Qwen3.5-9B executor match the primary protocol, preventing any adapted state from entering the frozen reference.

Running task. To make the protocol differences concrete, all flows below reuse the same ALFWorld failure introduced in Appendix C.1: the agent must place the salt shaker beside the coffee machine in

drawer 2, but the default executor places it in drawer 1 and repeatedly inspects the wrong drawer. Each method therefore receives the same task setting and execution evidence; what differs is the state it is allowed to update.

Dynamic Cheatsheet. Dynamic Cheatsheet processes the stream once and immediately updates a bounded textual cheatsheet after each task, using the question and answer/interaction trace without verifier labels, rewards, or reference answers; the revised sheet is available to the next task.

TASK : place the salt shaker beside the coffee machine in drawer 2
⇓
EXECUTION : default executor places it in drawer 1 and repeats recovery actions
⇓
MEMORY INPUT : task text + interaction trace, without verifier labels or rewards
⇓
DC UPDATE : Curator rewrites the bounded textual cheatsheet
⇓
WHEN USED : the revised cheatsheet is available to the next task.

ACE. In our evaluation, ACE uses a single-pass online protocol. After each batch, it applies the Reflector and Curator sequentially over the completed samples. The Reflector updates helpful/harmful/neutral counts for existing bullets, while the Curator applies localized ADD operations only. The resulting context affects the next batch and is not used to regenerate an answer for the same sample. Here, "single-pass" describes our evaluation configuration rather than the full adaptation scope supported by ACE.

TASK : the same salt-shaker placement task and failed execution
⇓
BATCH EVIDENCE : the completed samples in the current batch are processed in order
⇓
REFLECTOR : update helpful, harmful, and neutral counts for existing bullets
⇓
CURATOR : add localized experience bullets and prune to the context limit
⇓
WHEN USED : the revised ACE context begins with the next batch.

Thus, DC updates before the next task, whereas ACE updates only after the completed batch.

The reported internal controls differ only in the permitted adaptation surface. Harness-only evolves components and their exact-assembly playbooks while fixing the executor; Model-only retains the initial harness and adapts only model routing; and Full enables both. The four ablations remove native-peer evidence, lifecycle transitions, or KPI feedback, or allow every schema-valid candidate to bypass the counterfactual Gate. For these four lightweight variants, all settings other than the named intervention and isolated run root are inherited from Full.

G.2 ADAPTATION-SURFACE CONTROL DETAILS

Harness-only. This control retains the same LLM Weaver as Full, but its task-executor allow list contains only local Qwen3.5-9B. The Evolver is restricted to the harness surface: it may add, modify, or retire components, and exact-assembly playbooks may be induced by successful executions or Validator-selected assemblies. Model-routing and joint operations are disabled. Failure attribution, source/native-peer validation, the resource Gate, KPI feedback, and lifecycle promotion, demotion, and retirement remain enabled. The control therefore measures improvement from component composition and persistent harness assets under a fixed executor.

TASK EVIDENCE : wrong drawer selection followed by a repeated recovery loop

$\Downarrow$

FIXED EXECUTOR : Qwen3.5-9B cannot be replaced

$\Downarrow$

CANDIDATE SEARCH : revise execution-control components or their exact assembly

$\Downarrow$

VALIDATION : replay the harness candidate on the source and matched peer

$\Downarrow$

PERMITTED UPDATE : component and playbook state only.

Model-only. This control uses the fixed initial-harness strategy: the initial component set cannot be added to, modified, or retired, and runtime playbook induction is disabled. The Evolver is restricted to the model surface and may create scoped model-routing rules, but it cannot propose component or joint operations. Candidate executors come from the same model pool as Full and remain subject to the same source/native-peer validation, resource Gate, and lifecycle management. The control therefore measures improvement obtainable from task-conditioned executor selection alone.

TASK EVIDENCE : the same wrong placement and repeated recovery loop

$\Downarrow$

FIXED HARNESS : the initial components cannot be changed or composed into new playbooks

$\Downarrow$

CANDIDATE SEARCH : route this failure scope to another executor

$\Downarrow$

VALIDATION : replay the routed model on the source and matched peer

$\Downarrow$

PERMITTED UPDATE : a scoped model-routing rule only.

Full. Full HarnessWeaver retains all three adaptation surfaces. After mechanism attribution, it compares model-only, harness-only, and joint candidates under the same evidence and resource constraints, then commits the highest-ranked admissible update.

TASK EVIDENCE : the same wrong placement and repeated recovery loop

$\Downarrow$

ATTRIBUTION : wrong-goal choice and repetitive recovery form one mechanism

$\Downarrow$

HARNESS CANDIDATE : revise execution control

MODEL CANDIDATE : change the executor

JOINT CANDIDATE : change both surfaces

$\Downarrow$

PAIRED VALIDATION : compare source, native-peer, and resource evidence

$\Downarrow$

SELECTED UPDATE : commit the highest-ranked admissible intervention.

Shared protocol. Both controls use the same ALFWorld evolution/development/peer partitions, ordered batch plan, frozen baselines, five rounds per batch, seed, task prompt, environment interface, and verifier as Full. Development remains read-only and validation replays do not enter the primary success metric. The controls retain the same execution and validation backbone; only the failure attribution, candidate surface, and persistent state that may change differ. The paper reports the completed ALFWorld controls used for the adaptation-surface comparison.

G.3 COST-ORIENTED LIGHTWEIGHT VARIANTS

Each variant modifies one mechanism in full HarnessWeaver. The descriptions below specify the disabled functionality and the retained execution path; token savings are measured outcomes rather than guaranteed properties of a configuration.

Full admission reference. Full first checks candidate structure, then replays the candidate on its failure sources and matched native peers before assigning an evidence tier and lifecycle state.

RUNNING CANDIDATE : an evidence-linked repair for the salt-shaker failure
⇓
ENGINEERING CHECK : schema, type, dependency, and model-availability checks
⇓
EFFECT CHECK : source replay + matched native-peer replay
⇓
ADMISSION : assign the supported evidence tier and lifecycle state.

w/o Native Peer. Native-peer selection and execution are disabled, while candidate validation on the failed source tasks is retained. Positive source recovery can provide `source_only` PROBATION evidence; it is not independent peer support and does not by itself qualify a candidate for ACTIVE admission. Proposal generation, KPI feedback, and subsequent lifecycle management remain enabled.

SAME CANDIDATE : the salt-shaker repair passes engineering checks
⇓
AVAILABLE EVIDENCE : recovery is tested on the failed source task only
⇓
ADMISSION : record source-only evidence and enter controlled PROBATION
⇓
LATER USE : real task outcomes can still drive lifecycle updates.

w/o Lifecycle. Source/peer validation and candidate admission checks are retained, but accepted candidates are forced directly into ACTIVE. Automatic promotion, demotion, and retirement for non-use based on subsequent usage statistics are disabled. This removes ongoing status management while retaining proposal generation and validation. KPIs remain recorded and available to the Weaver for selection.

SAME CANDIDATE : the salt-shaker repair is checked on source and native peer
⇓
ADMISSION : a passing candidate is written directly as ACTIVE
⇓
LATER USE : no automatic promotion, demotion, or idle retirement.

w/o KPI Feedback. KPI feedback to the Weaver and model-pool statistics injection are disabled. Historical performance statistics are withheld from the component, routing-rule, playbook, and model-selection context, and the associated KPI-based ranking is disabled. Task execution, outcome logging, failure attribution, source/peer validation, and lifecycle management remain enabled. The intervention removes selection-time statistical feedback rather than evaluation records or all task feedback.

SAME CANDIDATE : the salt-shaker repair receives the full validation Gate

$$\Downarrow$$

PERSISTENT STATE : the accepted asset retains normal lifecycle management

$$\Downarrow$$

SUBSEQUENT SELECTION : the Weaver sees no historical model or asset KPI context.

Accept-All. Source/peer outcome replay and the performance-based admission Gate are bypassed, and engineering-valid candidates are admitted directly as ACTIVE. Invalid schema or type operations remain rejected; components must pass a deterministic build check, and routing or joint candidates must reference a selectable model. Proposal generation and subsequent lifecycle management remain enabled. Reduced validation overhead therefore comes with weaker initial performance screening.

SAME CANDIDATE : the salt-shaker repair is proposed from the same failure

$$\Downarrow$$

ENGINEERING CHECK : schema, type, build, and model availability only

$$\Downarrow$$

SKIPPED STEP : no source or native-peer outcome replay

$$\Downarrow$$

ADMISSION : an engineering-valid candidate enters ACTIVE directly.

Thus the variants intervene locally at different stages: evidence acquisition, persistent-state transitions, selection-time feedback, or initial performance screening.