Living-Harness IS AN INTERACTIVE-AGENT EVOLVER

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents may recover from a failure within an episode or after a retry, yet the same execution failure can recur in later tasks because post-episode feedback rarely revises the persistent harness that guides future interactions. Static harnesses improve reliability through fixed tools, context, memory, and workflow structures, but remain unchanged after deployment. We propose **Living-Harness**, a self-evolving agent harness that converts each completed trajectory and its evaluator signals into posterior evidence for bounded harness updates. Guided by a domain-level **Evolution-SOP** (**S**tandard **O**perating **P**rocedure), Living-Harness extracts an episode abstraction and structured update evidence, and writes two complementary forms of procedural knowledge: episodic memory that records trigger conditions, failure patterns, and recovery actions, and a state graph that records state nodes, repair edges, and transition rules. The updated harness state is retrieved to guide future interactions, while tools and base context remain frozen, allowing procedural repairs to accumulate across evolution cycles. On eight interactive environments derived from τ^2-Bench and MultiWOZ-2.4, Living-Harness improves average Pass@1 over the strongest interactive baseline by 10.07 and 9.91 percentage points, respectively, and supports retrieval-only reuse of the evolved harness state across model backbones.

1 INTRODUCTION

Large language model (LLM) agents are increasingly deployed as decision-making interfaces for task-oriented dialogue and tool-mediated environments (Wang et al., 2024b; Xi et al., 2025; Budzianowski et al., 2018; Yao et al., 2024). Although an agent may recognize an error from feedback or recover after a task-local retry, the same execution failure can recur in later interactions because the correction often disappears with the completed episode. For example, an agent may correctly conclude that a user should be transferred to a human operator, yet still fail to invoke the required transfer tool when a similar situation appears again. This exposes a gap between correcting an individual response and repairing the persistent procedure that guides future behavior. Practical agent systems address procedural reliability through an external *harness*, which organizes prompts, tools, context, memory, workflows, and evaluation interfaces around the base model (Meng et al., 2026; Lee et al., 2026). However, as illustrated in Figure 1, a static harness is fixed after deployment: it can impose predefined rules, but it cannot incorporate newly observed failure patterns and recovery actions into its future procedure. This leads to a concrete question: *how can an agent turn post-episode failures into persistent procedural repairs?*

Existing methods address parts of this problem from two directions. Harness-design and workflow-optimization methods construct stronger prompts, SOPs, tool constraints, skills, or execution structures that keep an agent within predefined paths (Meng et al., 2026; Lee et al., 2026; Ye et al., 2025; Xiong et al., 2025; Zhang et al., 2025b). They demonstrate the value of external procedural structure, but the resulting harness is commonly designed or optimized before deployment and then reused as a fixed artifact. Reflection and experience-memory methods instead retain critiques, summaries, or successful and failed trajectories for subsequent attempts (Shinn et al., 2023; Zhao et al., 2024; Ouyang et al., 2025). These methods provide useful experience, but a textual lesson such as "transfer the user to a human" does not by itself specify the trigger condition, required tool action, and workflow transition needed to prevent the same failure in future episodes. The missing capability is therefore not feedback collection alone, but *persistent procedural repair*: converting

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

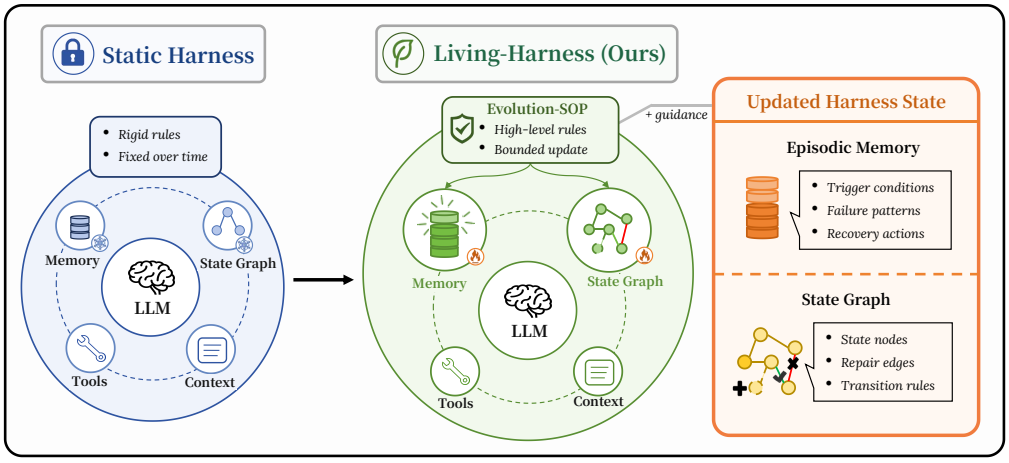


Figure 1: Static vs. living harnesses. A static harness remains fixed after deployment. Living-Harness uses an Evolution-SOP to guide bounded updates to episodic memory and the state graph for future retrieval, while tools and base context remain frozen.

evaluator-grounded failure evidence into a reusable relation between when a failure occurs, what action or transition is missing, and how a later rollout should recover.

A useful procedural-repair mechanism must satisfy several requirements. First, an update should be grounded in a completed trajectory and its evaluation signals, rather than in an unverified self-critique alone. Second, the repair must persist across episodes while remaining scoped to the relevant task family or failure condition. Third, adaptation should modify only the stateful parts of the harness, leaving the available tools and base context frozen so that accumulated experience does not freely rewrite the agent's operational boundaries. Finally, the persistent state should capture two complementary forms of knowledge: *experiential knowledge* about why a failure occurred and how it was recovered, and *workflow knowledge* about which state-conditioned action or transition should be introduced or revised. These requirements motivate an evolving harness state composed of complementary episodic memory and state-graph structures.

We therefore propose **Living-Harness**, a self-evolving harness that converts evaluated interaction trajectories into persistent procedural repairs. As shown in Figure 1, the tools and base context remain frozen, while episodic memory and the state graph evolve. After each episode, an evaluator provides feedback on the completed interaction, and a domain-level **Evolution-SOP** guides how failures are interpreted and which repairs are committed to the harness state. Episodic memory records trigger conditions, failure patterns, and recovery actions, while the state graph records state nodes, repair edges, and transition rules. In future episodes, relevant memory and graph entries are retrieved as procedural context, allowing recurring failures to progressively improve subsequent interactions.

To formalize this cross-episode adaptation, we view the agent as reasoning under uncertainty not only about the environment, but also about the procedural knowledge available to future decisions. This program-state POMDP perspective distinguishes within-episode execution from post-episode harness revision and motivates our rollout–evaluate–update formulation.

The main contributions of this paper are summarized as follows:

- **Persistent procedural repair for interactive agents.** We identify a gap between task-local correction and cross-episode adaptation: feedback may improve an individual retry without installing a reusable repair into the harness that guides later interactions. We formulate the target of adaptation as evaluator-grounded procedural repairs that persist beyond the episode in which a failure is observed.
- **The Living-Harness framework.** We introduce a rollout–evaluate–update framework in which a domain-level Evolution-SOP converts completed trajectories and evaluation signals into bounded updates of episodic memory and a state graph. These components store complementary experi-

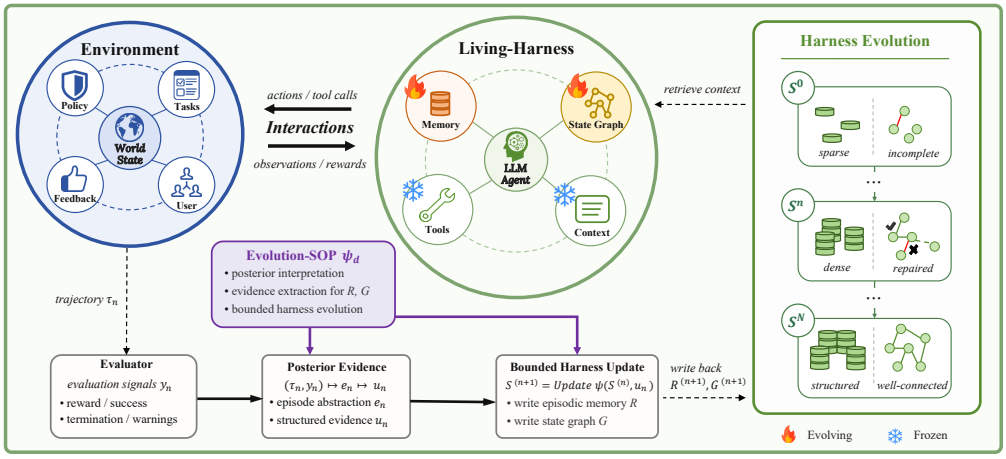


Figure 2: Overview of Living-Harness. In episode n, the agent interacts with the environment under harness state $S^{(n)}$, producing trajectory τ_n and evaluation signals y_n. Guided by the Evolution-SOP ψ_d, these signals are converted into posterior evidence and used to update episodic memory $\mathcal{R}$ and the state graph G. The resulting state $S^{(n+1)}$ is written back and retrieved to guide future episodes, while tools and base context remain frozen.

ential and workflow knowledge, are retrieved to guide future episodes, and evolve while the tools and base context remain frozen. We further provide a program-state POMDP interpretation of this cross-episode update process.

- **Evaluation of recovery accumulation and harness reuse.** We evaluate Living-Harness on eight interactive environments derived from τ^2-Bench and MultiWOZ-2.4. It improves average Pass@1 over the strongest interactive baseline by 10.07 and 9.91 percentage points, respectively, while exhibiting cycle-wise recovery accumulation and retrieval-only reuse of the evolved harness state across model backbones.

2 Related Work

Interactive LLM Agents. Research on interactive LLM agents examines how language models act over multiple turns by conditioning on observations, user feedback, tool outputs, and task constraints rather than producing isolated one-shot responses. Task-oriented dialogue benchmarks provide an early formulation of this setting, requiring agents to maintain dialogue state, follow domain policies, and complete user goals, as formalized by MultiWOZ (Budzianowski et al., 2018; Ye et al., 2022) and Schema-Guided Dialogue (Rastogi et al., 2020); recent tool-agent-user benchmarks extend it to realistic domains requiring both tool use and policy compliance (Yao et al., 2024). LLM agents broaden this line to reasoning, tool use, coordination, and trial-level adaptation, as summarized in recent surveys on LLM-based autonomous agents (Wang et al., 2024b; Xi et al., 2025). Representative systems include WebGPT (Nakano et al., 2021) for browsing-grounded answering, ReAct (Yao et al., 2023) for reasoning-action interleaving, Toolformer (Schick et al., 2023) and ToolLLM (Qin et al., 2024) for API use, AutoGen (Wu et al., 2023) for multi-agent coordination, and Reflexion (Shinn et al., 2023) for feedback-based retry. Together, these methods establish interaction and feedback as useful adaptation signals. Living-Harness focuses on how evaluated episodes revise the procedural state retrieved by later tasks.

Agent Harnesses. As LLM agents move from isolated prompting to tool use, memory, evaluation, and multi-step execution, their behavior is increasingly shaped by the external program surrounding the model. We use *agent harness* to denote this control layer, which externalizes prompts, tools, memory, workflows, execution loops, constraints, tracing, and evaluation interfaces (Meng et al., 2026; Lee et al., 2026). Studies of agent evaluation (Kapoor et al., 2025), context engineering (Ye et al., 2026), meta-tooling (Abuzakuk et al., 2026), workflow optimization (Hu et al., 2025; Zhang et al., 2025b), guardrails, sessions, tracing, and skill (Li et al., 2026; Li, 2026; Xu et al., 2026; Lu et al., 2026) benchmarks further show that this surrounding structure strongly affects practical

reliability. Skills are a central harness-level abstraction because they package procedural knowledge into reusable, invocable units, including instructions, code, resources, applicability conditions, and interfaces that can be loaded or selected at inference time (Xu & Yan, 2026; Jiang et al., 2026; Ling et al., 2026; Wang et al., 2024a). Existing harness and skill systems provide reusable procedural structure, but are commonly specified or optimized before deployment. Living-Harness instead updates its episodic memory and state graph from evaluated post-episode evidence while keeping tools and base context fixed.

Self-Evolving Agent Systems. Self-evolving agent systems turn interaction experience into persistent memory, skills, workflows, and other procedural artifacts (Gao et al., 2026; Fang et al., 2025; Liang et al., 2026). Prior work develops reusable reasoning memories (Zhao et al., 2024; Ouyang et al., 2025), skill libraries and routing mechanisms (Alzubi et al., 2026; Wang et al., 2024a; Xia et al., 2026; Zhang et al., 2026; Wang et al., 2026a), and evolving workflows, policies, architectures, environments, and test-time strategies (Zhang et al., 2025b; Yang et al., 2026; Zhang et al., 2025a; Dong et al., 2026; Wang et al., 2026b). Reflexion (Shinn et al., 2023) stores verbal self-reflections in episodic memory for subsequent trials. Agent Workflow Memory (AWM) (Wang et al., 2025) induces reusable workflows offline from examples or online from trajectories judged successful by an evaluator. Living-Harness jointly updates episodic memory and a state graph from evaluated failures, guided by an Evolution-SOP under fixed execution constraints.

3 METHOD

This section presents **Living-Harness**, a self-evolving agent harness that accumulates persistent procedural repairs through a rollout–evaluate–update loop. As illustrated in Figure 2, the agent retrieves relevant entries from the current harness state and interacts with the environment under fixed tools and base context. After the episode is evaluated, the **Evolution-SOP** converts the completed interaction and its evaluation signals into structured evidence for updating episodic memory and the state graph. The updated state is written back for retrieval in later episodes, allowing procedural repairs to accumulate across interactions.

3.1 PROBLEM SETTING AND HARNESS STATE

We consider an LLM agent operating in a domain d over a sequence of interaction episodes. In episode n, the agent receives a task x_n, interacts with the environment, and produces a trajectory τ_n consisting of observations, responses, and tool actions. After the interaction terminates, an evaluator produces signals y_n describing its execution outcome. The goal is to use (τ_n, y_n) to improve the procedural guidance available to later episodes.

Living-Harness separates fixed execution resources from an evolving procedural state. At episode n, the harness is represented as

$$H_d^{(n)} = \left(C_d^{\mathrm{act}}, \psi_d, S_d^{(n)}\right), \qquad S_d^{(n)} = \left(\mathcal{R}_d^{(n)}, G_d^{(n)}\right). \tag{1}$$

Here, C_d^{act} contains the tools, base context, and domain rules available to the actor; ψ_d is the fixed Evolution-SOP governing post-episode updates; and $S_d^{(n)}$ is the evolving harness state. The state remains fixed throughout the current episode and may be revised only after the completed interaction has been evaluated. Living-Harness therefore updates neither the base model nor C_d^{act}; its adaptation target is $S_d^{(n)}$.

The evolving state contains two complementary forms of procedural knowledge. Episodic memory $\mathcal{R}_d^{(n)}$ stores scoped records of trigger conditions, failure patterns, and recovery actions. The state graph $G_d^{(n)}$ stores state nodes, transition rules, and repair edges that connect observed conditions to missing or revised actions. During later episodes, relevant entries from both components are retrieved and rendered as procedural context, while the fixed rules in C_d^{act} retain precedence.

Conceptually, $S_d^{(n)}$ induces an episode-level program state $z^{(n)}$: the environment evolves within an episode, whereas the procedural information in $z^{(n)}$ changes only through post-episode harness updates. This two-timescale view is formalized in the Program-State Formulation.

Initialization. The initial state contains no episodic memory and only a coarse graph scaffold:

$$S_d^{(0)} = (\emptyset, G_{d,\mathrm{scaf}}) . \tag{2}$$

The scaffold provides domain roots and task-family structure but contains no fine-grained failure repairs. Such repairs are induced from evaluated interaction experience through the Evolution-SOP.

3.2 Evolution-SOP for Procedural Repair

The **Evolution-SOP** ψ_d is a fixed, domain-level protocol that governs how evaluated interactions revise the evolving harness state. It operates after rollout rather than serving as part of the actor's task-execution prompt. Across domains, Living-Harness uses the same posterior–extract–commit procedure, while ψ_d provides domain-specific guidance for interpreting failures, assigning update scope, and respecting domain and tool constraints. Adapting the framework to a new domain retains the update pipeline and state schemas while specifying the domain's tools, constraints, and scope rules in C_d^{act} and ψ_d.

Given a task x_n, its completed trajectory τ_n, evaluator signals y_n, and the fixed execution resources C_d^{act}, the Evolution-SOP first produces an episode abstraction:

$$e_n = \mathrm{Post}_{M,\psi_d}\left(x_n, \tau_n, y_n, C_d^{\mathrm{act}}\right) . \tag{3}$$

The abstraction identifies the task objective, verified interaction facts, execution outcome, and the critical failure or recovery point. Rather than retaining the full trajectory, it isolates the procedural information that may remain useful after the current episode ends.

The episode abstraction is then converted into evidence for the two components of the harness state:

$$u_n = \left(u_n^{\mathcal{R}}, u_n^{G}\right) = \mathrm{Extract}_{\psi_d}(e_n). \tag{4}$$

Here, $u_n^{\mathcal{R}}$ captures trigger conditions, failure patterns, and recovery actions for episodic memory, while u_n^{G} captures the corresponding states, actions, and transitions for the state graph. Thus, the memory preserves why and under what condition a repair is useful, whereas the graph records where that repair changes the future procedure.

The extracted evidence is treated as a candidate repair. Before being committed, it is checked for evidential support, task scope, and consistency with the fixed domain and tool constraints. Accepted candidates create or strengthen memory entries and state-conditioned repair edges; unsupported or conflicting candidates leave the persistent state unchanged.

3.3 Rollout–Evaluate–Update Loop

At episode n, Living-Harness constructs a task-conditioned query from the current task x_n and its scope f_n:

$$q_n = Q(x_n, f_n). \tag{5}$$

Relevant entries are retrieved from the current episodic memory and state graph and rendered as actor-facing procedural context:

$$\kappa_n = \mathrm{Render}\Big(\mathrm{Retrieve}_{\mathcal{R}}(\mathcal{R}^{(n)}, q_n), \mathrm{Retrieve}_{G}(G^{(n)}, q_n)\Big) . \tag{6}$$

This selective projection exposes the procedural knowledge most relevant to the current task rather than replaying the complete interaction history. The agent then interacts with the environment using the fixed execution resources and the retrieved context:

$$\tau_n \sim p_M\big(\tau \mid x_n, C_d^{\mathrm{act}}, \kappa_n\big) . \tag{7}$$

The harness state remains fixed throughout the rollout.

After the interaction is completed, the evaluator produces

$$y_n = E(x_n, \tau_n). \tag{8}$$

The Evolution-SOP converts τ_n and y_n into the episode abstraction e_n and candidate repair evidence u_n defined above. The accepted evidence is then written into the persistent harness state:

$$S^{(n+1)} = \mathrm{Update}_{\psi_d}\left(S^{(n)}, u_n; C_d^{\mathrm{act}}\right) . \tag{9}$$

Table 1: Main results on τ^2-Bench and MultiWOZ-2.4. All values are Pass@1 (%). Averages are weighted by the number of evaluated tasks. Bold and underlined values indicate the best and second-best results, respectively.

	τ^2-Bench				MultiWOZ-2.4			
Model	Retail	Airline	Telecom	Average	1-Domain	2-Domains	3-Domains	Average
Flagship Models								
Gemini 3 Pro	75.88	80.50	**91.01**	<u>82.92</u>	**79.20**	54.52	<u>24.48</u>	<u>55.80</u>
GLM-5	73.68	<u>82.50</u>	<u>86.84</u>	80.66	46.02	21.24	0.00	23.80
Qwen3-max	72.20	59.50	84.20	74.85	37.17	35.97	0.00	31.10
GPT-5.2	57.02	70.00	52.23	57.39	48.23	37.88	7.69	35.90
Kimi-k2	70.60	56.50	65.80	66.11	54.88	41.56	0.00	38.63
Interactive Baselines (GPT-5.2)								
ReAct	52.63	66.00	53.86	55.54	70.35	19.18	9.09	29.30
Reflexion	<u>78.07</u>	80.00	64.91	73.02	<u>77.88</u>	51.35	21.68	53.10
AWM	56.14	66.00	39.47	51.08	67.26	40.57	13.99	42.80
ReasoningBank	59.65	70.00	37.72	52.52	77.27	<u>58.33</u>	7.50	55.59
EvoTest	57.31	64.00	38.07	50.62	62.39	34.86	11.89	37.80
Ours (GPT-5.2)								
Living-Harness	**85.96**	**88.00**	78.07	**83.09**	76.55	**70.52**	**25.87**	**65.50**

The memory update preserves reusable failure–recovery experience, while the graph update revises the corresponding procedural transitions. If no candidate repair is accepted, the state remains unchanged. Otherwise, the updated state becomes available only to subsequent interactions, completing the rollout–evaluate–update loop without allowing post-episode evidence to alter the rollout that produced it.

Task-local correction and persistent evolution. For tasks that permit retries, a failed attempt may produce a local reflection used only within the current task instance. Such task-local feedback is discarded when the instance ends. Persistent evolution instead commits evaluator-grounded repairs to episodic memory and the state graph, making them available to later task instances.

3.4 Program-State Analysis

We analyze Living-Harness through a Partially Observable Markov Decision Process (POMDP) lens. In a standard POMDP, the environment state is not directly observed and must instead be inferred from the interaction history. Living-Harness augments this view with an episode-level program state representing the procedural information available from the evolving harness.

Definition 1 (Environment-state belief). Let $s_t \in \mathcal{S}$ denote the environment state at interaction step t, and let $h_t = (o_{\le t}, a_{<t})$ denote the interaction history. The standard POMDP belief over the environment is

$$b_t(s) = P(s_t = s \mid h_t)\,. \tag{10}$$

Definition 2 (Program-augmented belief). Using the program state $z^{(n)}$ induced by the current harness state $S^{(n)}$, we augment the environment state and define the corresponding joint belief as

$$\widetilde{s}_t^{(n)} = \left(s_t, z^{(n)}\right), B_t^{(n)}(s,z) = P\left(s_t = s, z^{(n)} = z \mid h_t\right). \tag{11}$$

The environment-only belief is recovered by marginalization, $b_t(s) = \sum_z B_t^{(n)}(s,z)$.

The two states evolve at different timescales. During episode n, $z^{(n)}$ remains fixed while the joint belief is updated as

$$B_{t+1}^{(n)}(s',z) \propto \Omega(o_{t+1} \mid s') \sum_s T(s' \mid s, a_t, z) B_t^{(n)}(s,z). \tag{12}$$

Table 2: Per-cycle performance of Living-Harness. Parenthesized values report absolute changes from the preceding cycle. Rest. and Attr. denote Restaurant and Attraction. Bold and underlined values indicate the best and second-best cycle performance within each domain, respectively.

Cycle	τ^2-Bench			MultiWOZ-2.4				
	Retail	Airline	Telecom	Restaurant	Hotel	Train	Attraction	Taxi
0	57.02 (+0.00)	70.00 (+0.00)	57.39 (+0.00)	30.89 (+0.00)	24.11 (+0.00)	43.43 (+0.00)	40.15 (+0.00)	8.21 (+0.00)
1	74.56 (+17.54)	76.00 (+6.00)	67.54 (+10.15)	<u>43.47</u> (+12.58)	42.64 (+18.53)	64.65 (+21.22)	61.36 (+21.21)	28.72 (+20.51)
2	<u>79.82</u> (+5.26)	**88.00** (+12.00)	<u>71.92</u> (+4.38)	41.65 (-1.82)	<u>46.95</u> (+4.31)	<u>65.86</u> (+1.21)	<u>63.89</u> (+2.53)	<u>36.41</u> (+7.69)
3	**85.96** (+6.14)	<u>86.00</u> (-2.00)	**78.07** (+6.15)	**44.85** (+3.20)	**48.22** (+1.27)	**66.26** (+0.40)	**65.40** (+1.51)	**36.92** (+0.51)

After evaluation, updates to episodic memory and the state graph induce a new program state for later episodes. Thus, t indexes within-episode inference, while n indexes cross-episode harness evolution.

Proposition 1 (Information refinement). For the nested information sets $\mathcal{I}_s = \sigma(h_t) \subseteq \mathcal{I}_{s,z} = \sigma(h_t, z^{(n)})$, consider an execution-relevant latent variable $\xi \in L^2$. Define $\mathcal{B}(\mathcal{I}) = \mathbb{E}[\mathrm{Var}(\xi \mid \mathcal{I})]$. Then $\mathcal{B}(\mathcal{I}_{s,z}) = \mathcal{B}(\mathcal{I}_s) - \Gamma_z \leq \mathcal{B}(\mathcal{I}_s)$, where $\Gamma_z = \mathbb{E}[\mathrm{Var}(\mathbb{E}[\xi \mid \mathcal{I}_{s,z}] \mid \mathcal{I}_s)] \geq 0$. The inequality is strict when $\Gamma_z > 0$, that is, when the refined posterior mean is not $\mathcal{I}_s$-measurable.

Here, $\mathcal{B}(\mathcal{I})$ denotes the Bayes error bound under information $\mathcal{I}$, while $\mathcal{I}_{s,z}$ augments the interaction history with the retrieved memory and graph context represented by $z^{(n)}$. Proposition 1 therefore shows that informative procedural context can reduce the optimal prediction error relative to interaction history alone. In the implemented loop, extraction selects candidate procedural information, commit checks assess its support and scope, and retrieval supplies it to later decisions. Appendix A relates the benefit of this information to the approximation error in extracting and using it.

4 Experiments

4.1 Experimental Setup

Evaluation Benchmarks. We evaluate Living-Harness on two interactive benchmarks:

- **τ^2-Bench** (Barrès et al., 2025): a realistic benchmark for conversational agents with multi-turn interaction, policy constraints, and executable tool use. We use three domains: *Retail*, *Airline*, and *Telecom*.
- **MultiWOZ-2.4** (Ye et al., 2022): a corrected multi-domain task-oriented dialogue benchmark. We use five primary domains: *Restaurant*, *Hotel*, *Train*, *Attraction*, and *Taxi*.

Across both benchmarks, we report task success rate, denoted as Pass@1, as the primary metric.

Baselines. We compare Living-Harness against two groups of baselines:

- *Flagship Models*: We evaluate strong proprietary and open-weight backbones, including GPT-5.2 (OpenAI, 2026), Gemini 3 Pro (Pichai et al., 2025), GLM-5 (GLM, 2026), Qwen3-max (Yang et al., 2025), and Kimi-k2 (Team, 2025). These models are evaluated without online harness evolution to measure the base capability of each agent backbone.
- *Interactive Baselines*: We compare against representative interactive and self-improving agent methods, including ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), Agent Workflow Memory (AWM) (Wang et al., 2025), ReasoningBank (Ouyang et al., 2025), and EvoTest (He et al., 2025).

Implementation Details. All GPT-5.2-based interactive methods, including Living-Harness, use GPT-5.2 with medium reasoning effort as the backbone; within Living-Harness, the actor and all self-evolving modules share this same backbone. For τ^2-Bench, the simulated user is implemented

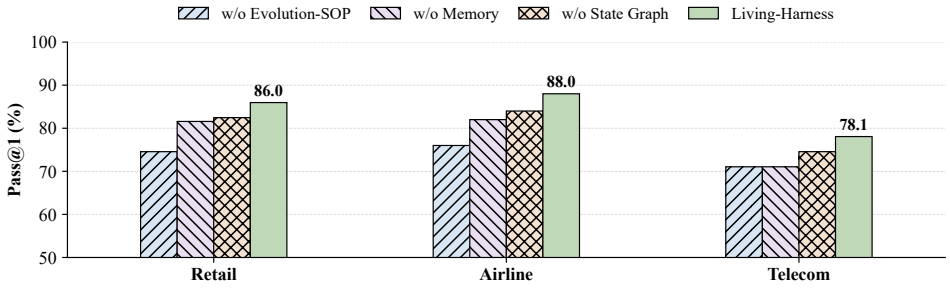


Figure 3: Component ablation on τ^2-Bench: removing the Evolution-SOP causes the largest drop, while memory and state graph ablations show complementary contributions.

Table 3: Retrieval-only cross-model transfer of the frozen Living-Harness state on MultiWOZ-2.4. Bold values denote scores obtained with the transferred harness state. Rest. and Attr. denote Restaurant and Attraction.

Model	Restaurant	Hotel	Train	Attraction	Taxi
Gemini 3 Pro	47.83	45.18	55.35	63.64	30.26
+ **Living-Harness**	**66.13**	**63.71**	**66.87**	**66.67**	**68.72**
GLM-5	24.94	16.50	18.18	27.27	0.00
+ **Living-Harness**	**39.36**	**38.83**	**38.38**	**38.13**	**43.08**
Qwen3-max	19.45	31.98	37.98	35.10	0.00
+ **Living-Harness**	**45.77**	**44.42**	**47.27**	**44.70**	**45.13**
Kimi-k2	40.27	35.37	38.42	37.01	0.00
+ **Living-Harness**	**44.85**	**43.65**	**47.27**	**47.22**	**45.13**

with GPT-5.1. Across the actor and all evolution modules, we use a sampling temperature of 0.2 and set nucleus sampling to $\texttt{top_p} = 1.0$. Where applicable, all interactive baselines use the same tool interface, evaluator, and task-level retry budget. We enable task-local reflexion with at most 3 trials per task; this local buffer is used only within the current instance and is never directly written into the global episodic memory $\mathcal{R}^{(n)}$ or state graph $G^{(n)}$. Both memory and graph retrieval use top-$k = 3$.

4.2 Main Results

Strong overall gains. Table 1 shows that Living-Harness achieves the highest overall Pass@1 on both τ^2-Bench (83.09) and MultiWOZ-2.4 (65.50). Using GPT-5.2 medium, it exceeds Gemini 3 Pro's 82.92 and Reflexion's 73.02 on τ^2-Bench. On MultiWOZ-2.4, it improves over ReasoningBank, the strongest interactive baseline, by 9.91 points and over Reflexion by 12.40 points. Its one-domain score remains slightly below Reflexion, but it leads the two- and three-domain groups, suggesting greater benefits for coordinating procedures across domains.

Self-evolution across cycles. Table 2 shows gains across evolution cycles. On τ^2-Bench, Retail rises from 57.02 to 85.96, Telecom reaches 78.07, and Airline peaks at 88.00 in Cycle 2 before declining to 86.00 in Cycle 3. On MultiWOZ-2.4, final-cycle gains over Cycle 0 range from 13.96 points on Restaurant to 28.71 on Taxi. The largest gains usually occur in the first evolution cycle, with smaller subsequent gains and occasional declines.

4.3 Ablation Studies

Ablation definitions. In *w/o Evolution-SOP*, we keep the memory and state graph containers but replace the domain update procedure with a generic extractor, disabling domain-specific commit gates and family-scoped update rules. In *w/o Memory*, episodic-memory retrieval and memory updates are disabled. In *w/o State Graph*, graph retrieval and updates are disabled.

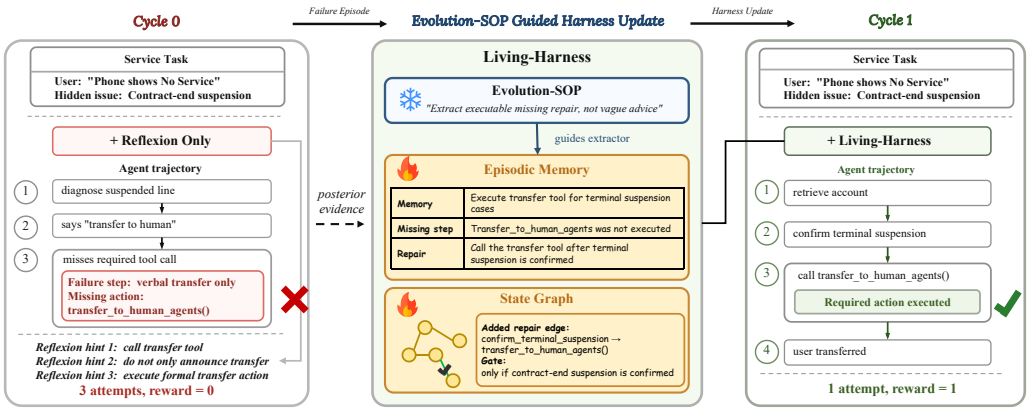


Figure 4: Persistent procedural repair. Reflexion recognizes the need for human assistance but omits the transfer call. Living-Harness converts the evaluated failure into episodic memory and a state-graph repair edge, enabling `transfer_to_human_agents()` execution and task completion in one attempt in Cycle 1.

Component ablation. Figure 3 shows that the full system performs best in every domain. Removing the Evolution-SOP causes the largest average drop, from 83.09 to 73.38, highlighting the contribution of structured posterior interpretation and bounded state updates. Removing memory or the state graph yields 77.34 and 79.50, respectively, supporting the complementary benefits of recovery lessons and workflow transitions.

4.4 In-depth Analysis

Table 3 evaluates cross-backbone reuse within the evaluated MultiWOZ-2.4 domains: the harness state evolved with GPT-5.2 is frozen and provided to other backbones. During target-model evaluation, the transferred backbone only retrieves from the frozen episodic memory and state graph, with no additional global updates. Across Gemini 3 Pro, GLM-5, Qwen3-max, and Kimi-k2, the frozen harness improves every reported domain score. The gains are especially large in domains where the base model struggles, such as Taxi, where GLM-5, Qwen3-max, and Kimi-k2 improve from 0.00 to 43.08, 45.13, and 45.13, respectively. The gains on Gemini 3 Pro further show that the stored procedural knowledge can complement a stronger backbone. Target actors retrieve the same frozen memory and graph without additional updates, demonstrating retrieval-only reuse of the evolved state across backbones.

Figure 4 illustrates how a retrieved repair can change later agent behaviors. In the failure cycle, Reflexion repeatedly identifies the correct high-level lesson: the agent should transfer the user to a human agent, but the rollout still fails because the required tool call is never executed. Living-Harness converts this failure into an executable repair: the missing action is stored both as episodic memory and as a state graph edge linking terminal suspension detection to `transfer_to_human_agents()`. The retrieved repair is not a vague advice but a state-conditioned repair that guides the required transfer action. This illustrates the key distinction from response-level reflection: recurring failures become retrievable repairs that shape future rollouts.

5 Conclusion

We introduced **Living-Harness**, which converts evaluated interaction failures into coordinated updates of episodic memory and workflow structure under an Evolution-SOP. The resulting harness state provides persistent, retrievable procedures for future tasks. Experiments on τ^2-Bench and MultiWOZ-2.4 demonstrate improved task performance, gains across evolution cycles, and retrieval-only reuse across backbones, supporting procedural adaptation through an evolving external harness.

AI USE STATEMENT

We used generative AI tools to assist with literature and citation checks, local editing to improve the clarity and readability of the manuscript, and LaTeX template migration and formatting. The authors reviewed and, where needed, revised the output at each stage, including the literature checks, wording changes, and formatting changes. The authors take full responsibility for the final content of this work, including its text, claims, citations, and artifacts.

REFERENCES

Sami Abuzakuk, Anne-Marie Kermarrec, Rishi Sharma, Rasmus Moorits Veski, and Martijn de Vos. Optimizing agentic workflows using meta-tools. *CoRR*, abs/2601.22037, 2026. doi: 10.48550/ARXIV.2601.22037. URL https://doi.org/10.48550/arXiv.2601.22037.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *CoRR*, abs/2603.02766, 2026. doi: 10.48550/ARXIV.2603.02766. URL https://doi.org/10.48550/arXiv.2603.02766.

Victor Barrès, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL https://doi.org/10.48550/arXiv.2506.07982.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 5016–5026, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1547. URL https://aclanthology.org/D18-1547/.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *CoRR*, abs/2604.18292, 2026. doi: 10.48550/ARXIV.2604.18292. URL https://doi.org/10.48550/arXiv.2604.18292.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems. *CoRR*, abs/2508.07407, 2025. doi: 10.48550/ARXIV.2508.07407. URL https://doi.org/10.48550/arXiv.2508.07407.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Trans. Mach. Learn. Res.*, 2026, 2026. URL https://openreview.net/forum?id=CTr3bovS5F.

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

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems. *CoRR*, abs/2510.13220, 2025. doi: 10.48550/ARXIV.2510.13220. URL https://doi.org/10.48550/arXiv.2510.13220.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

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

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Trans. Mach. Learn. Res.*, 2025, 2025. URL https://openreview.net/forum?id=Zy4uFzMviZ.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *CoRR*, abs/2603.28052, 2026. doi: 10.48550/ARXIV.2603.28052. URL https://doi.org/10.48550/arXiv.2603.28052.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *CoRR*, abs/2602.12670, 2026. doi: 10.48550/ARXIV.2602.12670. URL https://doi.org/10.48550/arXiv.2602.12670.

Xiaoxiao Li. When single-agent with skills replace multi-agent systems and when they fail. *CoRR*, abs/2601.04748, 2026. doi: 10.48550/ARXIV.2601.04748. URL https://doi.org/10.48550/arXiv.2601.04748.

Jiaqing Liang, Jinyi Han, Weijia Li, Xinyi Wang, Zhoujia Zhang, Zishang Jiang, Ying Liao, Tingyun Li, Ying Huang, Hao Shen, et al. Genericagent: A token-efficient self-evolving LLM agent via contextual information density maximization (V1.0). *CoRR*, abs/2604.17091, 2026. doi: 10.48550/ARXIV.2604.17091. URL https://doi.org/10.48550/arXiv.2604.17091.

George Ling, Shanshan Zhong, and Richard Huang. Agent skills: A data-driven analysis of claude skills for extending large language model functionality. *CoRR*, abs/2602.08004, 2026. doi: 10.48550/ARXIV.2602.08004. URL https://doi.org/10.48550/arXiv.2602.08004.

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

Qianyu Meng, Yanan Wang, Liyi Chen, Wei Wu, Yihang Li, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. Agent harness for large language model agents: A survey. 2026. doi: 10.20944/preprints202604.0428.v3. URL https://www.preprints.org/manuscript/202604.0428/v3.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021. URL https://arxiv.org/abs/2112.09332.

OpenAI. Openai gpt-5 system card, 2026. URL https://arxiv.org/abs/2601.03267.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *CoRR*, abs/2509.25140, 2025. doi: 10.48550/ARXIV.2509.25140. URL https://doi.org/10.48550/arXiv.2509.25140.

Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. A new era of intelligence with gemini 3. *Google. URL: https://blog.google/products-and-platforms/products/gemini/gemini-3/*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12,*

2020, pp. 8689–8696. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6394. URL https://doi.org/10.1609/aaai.v34i05.6394.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Kimi Team. Kimi K2: open agentic intelligence. *CoRR*, abs/2507.20534, 2025. doi: 10.48550/ARXIV.2507.20534. URL https://doi.org/10.48550/arXiv.2507.20534.

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. *Trans. Mach. Learn. Res.*, 2024, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Jiayu Wang, Yifei Ming, Zixuan Ke, Shafiq Joty, Aws Albarghouthi, and Frederic Sala. Skillorchestra: Learning to route agents via skill transfer. *CoRR*, abs/2602.19672, 2026a. doi: 10.48550/ARXIV.2602.19672. URL https://doi.org/10.48550/arXiv.2602.19672.

Junjie Wang, Yiming Ren, and Haoyang Zhang. From procedural skills to strategy genes: Towards experience-driven test-time evolution. *CoRR*, abs/2604.15097, 2026b. doi: 10.48550/ARXIV.2604.15097. URL https://doi.org/10.48550/arXiv.2604.15097.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6): 186345, 2024b. doi: 10.1007/S11704-024-40231-1. URL https://doi.org/10.1007/s11704-024-40231-1.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *CoRR*, abs/2308.08155, 2023. doi: 10.48550/ARXIV.2308.08155. URL https://doi.org/10.48550/arXiv.2308.08155.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: a survey. *Sci. China Inf. Sci.*, 68(2), 2025. doi: 10.1007/S11432-024-4222-0. URL https://doi.org/10.1007/s11432-024-4222-0.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *CoRR*, abs/2602.08234, 2026. doi: 10.48550/ARXIV.2602.08234. URL https://doi.org/10.48550/arXiv.2602.08234.

Weimin Xiong, Yifan Song, Qingxiu Dong, Bingchan Zhao, Feifan Song, Xun Wang, and Sujian Li. MPO: boosting LLM agents with meta plan optimization. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 3914–3935. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-EMNLP.210. URL https://doi.org/10.18653/v1/2025.findings-emnlp.210.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *CoRR*, abs/2602.12430, 2026. doi: 10.48550/ARXIV.2602.12430. URL https://doi.org/10.48550/arXiv.2602.12430.

Yangjie Xu, Lujun Li, Lama Sleem, Niccolò Gentile, Yewei Song, Yiqun Wang, Siming Ji, Wenbo Wu, and Radu State. Agent skill framework: Perspectives on the potential of small language models in industrial environments. *CoRR*, abs/2602.16653, 2026. doi: 10.48550/ARXIV.2602.16653. URL https://doi.org/10.48550/arXiv.2602.16653.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard H. Hovy. EVOTOOL: self-evolving tool-use policy optimization in LLM agents via blame-aware mutation and diversity-aware selection. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 43553–43572. Association for Computational Linguistics, 2026. URL https://aclanthology.org/2026.acl-long.2016/.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045, 2024. doi: 10.48550/ARXIV.2406.12045. URL https://doi.org/10.48550/arXiv.2406.12045.

Anbang Ye, Qianran Ma, Jia Chen, Muqi Li, Tong Li, Fujiao Liu, Siqi Mai, Meichen Lu, Haitao Bao, and Yang You. Sop-agent: Empower general purpose AI agent with domain-specific sops. *CoRR*, abs/2501.09316, 2025. doi: 10.48550/ARXIV.2501.09316. URL https://doi.org/10.48550/arXiv.2501.09316.

Fanghua Ye, Jarana Manotumruksa, and Emine Yilmaz. Multiwoz 2.4: A multi-domain task-oriented dialogue dataset with essential annotation corrections to improve state tracking evaluation. In Oliver Lemon, Dilek Hakkani-Tür, Junyi Jessy Li, Arash Ashrafzadeh, Daniel Hernández García, Malihe Alikhani, David Vandyke, and Ondrej Dusek (eds.), *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue, SIGDIAL 2022, Edinburgh, UK, 07-09 September 2022*, pp. 351–360. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.SIGDIAL-1.34. URL https://doi.org/10.18653/v1/2022.sigdial-1.34.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. *CoRR*, abs/2601.21557, 2026. doi: 10.48550/ARXIV.2601.21557. URL https://doi.org/10.48550/arXiv.2601.21557.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *CoRR*, abs/2602.02474, 2026. doi: 10.48550/ARXIV.2602.02474. URL https://doi.org/10.48550/arXiv.2602.02474.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025b. URL https://openreview.net/forum?id=z5uVAKwmjf.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: LLM agents are experiential learners. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pp. 19632–19642. AAAI Press, 2024. doi: 10.1609/AAAI.V38I17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

SUPPLEMENTARY MATERIAL

This supplementary document provides the formal proof, extended reproducibility details, additional harness-state analyses, prompt specifications, and an expanded discussion of limitations for the main paper. The main paper is self-contained; the material below provides supporting technical detail and extended results.

A PROOF OF PROGRAM-STATE BELIEF TIGHTENING

This section gives the formal statement and proof of Proposition 1 in the main paper. The result provides an information-theoretic justification for representing the execution-guiding procedural state z: adding an informative state variable refines the available information and therefore cannot increase the optimal Bayes estimation error. It does not claim that every approximate implementation of z must strictly improve performance; strict improvement requires that z contain nonzero conditional information about execution-relevant latent variables.

Setup. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. Let $\xi \in L^2(\Omega)$ denote an execution-relevant latent variable, such as the valid workflow branch, hidden failure mode, or next recoverable state. Let $\mathcal{I}_s \subseteq \mathcal{F}$ denote the information available to an environment-only belief, and let $\mathcal{I}_{s,z} \subseteq \mathcal{F}$ denote the refined information available after additionally modeling the procedural state z. We assume the natural information-nesting condition

$$\mathcal{I}_s \subseteq \mathcal{I}_{s,z}, \tag{13}$$

which means that the augmented belief can ignore z and recover the environment-only belief.

Formal statement. For any information set $\mathcal{I}$, define the Bayes mean-squared error bound

$$\mathcal{B}(\mathcal{I}) := \inf_{\widehat{\xi} \in L^2(\mathcal{I})} \mathbb{E}\Big[(\xi - \widehat{\xi})^2\Big], \tag{14}$$

where $L^2(\mathcal{I})$ is the set of square-integrable $\mathcal{I}$-measurable estimators.

Proposition A.1 (Formal version of Proposition 1). Under the setup above, the environment-only and program-augmented Bayes error bounds satisfy

$$\mathcal{B}(\mathcal{I}_{s,z}) = \mathcal{B}(\mathcal{I}_s) - \Gamma_z \leq \mathcal{B}(\mathcal{I}_s), \tag{15}$$

where

$$\Gamma_z = \mathbb{E}[\mathrm{Var}(\mathbb{E}[\xi \mid \mathcal{I}_{s,z}] \mid \mathcal{I}_s)] \geq 0. \tag{16}$$

Moreover, the inequality is strict whenever $\Gamma_z > 0$, equivalently when the refined posterior mean $\mathbb{E}[\xi \mid \mathcal{I}_{s,z}]$ is not $\mathcal{I}_s$-measurable.

Proof. For any information set $\mathcal{I}$, the Bayes estimator under squared loss is the conditional expectation $\mu_{\mathcal{I}} = \mathbb{E}[\xi \mid \mathcal{I}]$. Hence,

$$\mathcal{B}(\mathcal{I}) = \mathbb{E}\big[(\xi - \mu_{\mathcal{I}})^2\big] = \mathbb{E}[\mathrm{Var}(\xi \mid \mathcal{I})]\,. \tag{17}$$

Since $\mathcal{I}_s \subseteq \mathcal{I}_{s,z}$, the conditional-variance decomposition gives

$$\mathrm{Var}(\xi \mid \mathcal{I}_s) = \mathbb{E}[\mathrm{Var}(\xi \mid \mathcal{I}_{s,z}) \mid \mathcal{I}_s] \tag{18}$$

$$+ \mathrm{Var}(\mathbb{E}[\xi \mid \mathcal{I}_{s,z}] \mid \mathcal{I}_s)\,. \tag{19}$$

Taking expectations on both sides yields

$$\mathcal{B}(\mathcal{I}_s) = \mathcal{B}(\mathcal{I}_{s,z}) + \Gamma_z. \tag{20}$$

Rearranging gives

$$\mathcal{B}(\mathcal{I}_{s,z}) = \mathcal{B}(\mathcal{I}_s) - \Gamma_z \leq \mathcal{B}(\mathcal{I}_s), \tag{21}$$

because $\Gamma_z \geq 0$. The inequality is strict exactly when $\Gamma_z > 0$. Since a conditional variance is zero if and only if the conditioned random variable is measurable with respect to the conditioning sigma-algebra, $\Gamma_z > 0$ holds when $\mathbb{E}[\xi \mid \mathcal{I}_{s,z}]$ is not already determined by $\mathcal{I}_s$. This completes the proof.

Approximate implementation. The proposition concerns the optimal Bayes estimator under a refined information set. Living-Harness realizes this refinement approximately through retrieved procedural context, posterior abstraction, and bounded state updates.

Let $\widetilde{\mu}_{s,z}$ be an $\mathcal{I}_{s,z}$-measurable estimator satisfying

$$\mathbb{E}\big[(\widetilde{\mu}_{s,z} - \mu_{\mathcal{I}_{s,z}})^2\big] \le \epsilon_z. \tag{22}$$

Then

$$\mathbb{E}\big[(\xi - \widetilde{\mu}_{s,z})^2\big] \le \mathcal{B}(\mathcal{I}_s) - \Gamma_z + \epsilon_z. \tag{23}$$

Proof. Since $\widetilde{\mu}_{s,z} - \mu_{\mathcal{I}_{s,z}}$ is $\mathcal{I}_{s,z}$-measurable and $\mathbb{E}[\xi - \mu_{\mathcal{I}_{s,z}} \mid \mathcal{I}_{s,z}] = 0$, the cross term vanishes:

$$\begin{aligned}
\mathbb{E}\big[(\xi - \widetilde{\mu}_{s,z})^2\big] &= \mathbb{E}\big[(\xi - \mu_{\mathcal{I}_{s,z}})^2\big] && (24)\\
&\quad + \mathbb{E}\big[(\widetilde{\mu}_{s,z} - \mu_{\mathcal{I}_{s,z}})^2\big] && (25)\\
&\le \mathcal{B}(\mathcal{I}_{s,z}) + \epsilon_z && (26)\\
&= \mathcal{B}(\mathcal{I}_s) - \Gamma_z + \epsilon_z. && (27)
\end{aligned}$$

Thus, the approximate augmented estimator has a lower upper bound than the environment-only Bayes bound whenever $\epsilon_z < \Gamma_z$. More generally, relative to an approximate environment-only estimator with excess error ϵ_s, the augmented upper bound is lower whenever

$$\epsilon_z - \epsilon_s < \Gamma_z. \tag{28}$$

This condition states that the information gain contributed by z must exceed the additional approximation error introduced by representing and using it.

Decision-risk version. The same monotonicity holds for general decision risk. Let $\mathcal{A}$ be an action space and let $\ell(a, \xi)$ be a loss incurred by taking action a when the execution-relevant latent variable is ξ. For an information set $\mathcal{I}$, let $\Pi(\mathcal{I})$ denote all $\mathcal{I}$-measurable policies. Define the optimal Bayes risk

$$\mathcal{J}^\star(\mathcal{I}) = \inf_{\pi \in \Pi(\mathcal{I})} \mathbb{E}[\ell(\pi, \xi)]. \tag{29}$$

Since $\mathcal{I}_s \subseteq \mathcal{I}_{s,z}$, we have $\Pi(\mathcal{I}_s) \subseteq \Pi(\mathcal{I}_{s,z})$. Therefore,

$$\begin{aligned}
\mathcal{J}^\star(\mathcal{I}_{s,z}) &= \inf_{\pi \in \Pi(\mathcal{I}_{s,z})} \mathbb{E}[\ell(\pi, \xi)] && (30)\\
&\le \inf_{\pi \in \Pi(\mathcal{I}_s)} \mathbb{E}[\ell(\pi, \xi)] = \mathcal{J}^\star(\mathcal{I}_s). && (31)
\end{aligned}$$

Implication for Living-Harness. The variable z captures retrievable workflow structure, recoverable failure modes, and episodic lessons under frozen execution constraints. Proposition A.1 shows that, when z is informative, marginalizing it away leaves avoidable uncertainty: explicitly representing z contracts the Bayes error bound by Γ_z. The approximate result further shows that this benefit holds when the information gain from z exceeds the error introduced by extracting and using it. The practical algorithm approximates this interpretation through retrieval, extraction, and gated harness-state updates rather than exact Bayesian filtering.

B Reproducibility and Experimental Details

This section provides supporting implementation details for the main paper. Section B.1 specifies the score-before-update protocol, Section B.2 details retrieval and commit gates, and Section B.3 summarizes the shared baseline configuration.

B.1 Online Scoring and Update Protocol

For each episode n, the actor first performs rollout using only the harness state available before that episode, $S^{(n)} = (\mathcal{R}^{(n)}, G^{(n)})$, together with the frozen domain context C_d. The benchmark evaluator then computes the task score from the completed trajectory. Only after this score has been

recorded do the posterior generator, memory extractor, and workflow extractor process the trajectory and propose updates to the global harness state.

Therefore, evidence generated from an episode is never available when computing that episode's reported score. For τ^2-Bench, the global harness state is updated after each scored episode. For MultiWOZ-2.4, updates are synchronized every four scored episodes: all episodes in the synchronization window are first rolled out and scored, and their evidence is committed only afterwards. This prevents same-instance global-state reuse while allowing adaptation over the online interaction stream.

Task-local reflexion is handled separately from persistent harness adaptation. A local reflection buffer may guide retries within the current task instance, but it is discarded after that instance and is never directly written into the global episodic memory $\mathcal{R}^{(n)}$ or state graph $G^{(n)}$.

B.2 Retrieval, Update Normalization, and Commit Gates

At the beginning of each episode, Living-Harness retrieves context from episodic memory and the state graph. Unless otherwise specified, both retrieval modules use top-$k = 3$. Retrieval follows a same-family-first strategy: candidates from the same task family are prioritized, and cross-family candidates are used only when same-family evidence is insufficient. Candidate memories and graph fragments are ranked by semantic relevance to the current task query, task-family compatibility, and accumulated confidence or support.

Extractor outputs are normalized into schema-constrained JSON before commitment. Each update must specify its task family, trigger condition, failure pattern, proposed repair, and confidence signal. Malformed outputs receive one repair pass; an output that still violates the schema is discarded.

Global updates are committed through the Evolution-SOP-defined update pipeline. A candidate must pass the following gates:

- **Schema gate**: the update must satisfy the required memory or graph schema.
- **Scope gate**: the update is committed to the corresponding task family unless cross-family transfer is explicitly supported.
- **Evidence gate**: the update must be grounded in evaluator feedback, trajectory evidence, or repeated failure patterns.
- **Constraint gate**: the update must not override frozen domain policies, tool preconditions, or execution constraints in C_d.
- **Merge gate**: semantically similar updates are merged with existing entries using semantic hashes and accumulated confidence rather than inserted as conflicting duplicates.

For episodic memory, committed updates create or strengthen reusable failure–repair lessons. For the state graph, committed updates mainly merge or strengthen state-conditioned repair edges. These gates reduce the risk that noisy or misattributed evaluator signals become persistent state; they do not provide full rollback or regression testing.

B.3 Baseline Configuration and Cost Profile

All GPT-5.2-based interactive methods use GPT-5.2 with medium reasoning effort as the backbone. In Living-Harness, the actor and all evolution modules share this backbone. For τ^2-Bench, the simulated user is implemented with GPT-5.1 following the benchmark setting. All interactive baselines use the same tool interface, evaluator, and task-level retry budget whenever applicable.

For Reflexion, the agent is allowed at most three trials per task, matching the task-local reflection budget used by Living-Harness. ReAct follows its standard reasoning–acting loop under the same tool and interaction budget. For memory-based baselines such as AWM and ReasoningBank, retrieved memory is inserted into the actor context under the same context-budget constraint. Living-Harness couples post-episode abstraction with evidence, scope, and constraint checks before committing coordinated updates to episodic memory and the state graph.

For transfer experiments, target backbones receive retrieval-only access to the frozen source harness state, with no additional global updates during target-model evaluation.

B.4 API-Based Computing Infrastructure

All model inference was conducted through remote OpenAI-compatible API endpoints; no model was trained or hosted locally. Consequently, the provider-side GPU/CPU models, accelerator memory, operating system, and serving software were not exposed to us. The local runtime was used only for agent orchestration, tool execution, retrieval, harness-state persistence, and metric aggregation. We therefore report the model identifiers and inference configurations used in the experiments, while treating the provider-side computing infrastructure as unavailable.

C Additional Results

Harness-state scaling. Tables 4 and 5 report how the persistent harness state grows over evolution cycles. Cycle 0 denotes the empty pre-evolution state. For each domain, $|\mathcal{R}^{(n)}|$ is the number of episodic-memory entries after cycle n, while $|V_G^{(n)}|$ and $|E_G^{(n)}|$ denote the numbers of nodes and edges in the state graph. Parenthesized values report absolute increases over the preceding cycle. These measurements characterize state growth over the three evaluated cycles. Top-k retrieval limits the number of entries exposed to the actor, while long-run storage growth and retrieval latency require separate evaluation.

Table 4: Harness-state scaling on τ^2-Bench. Cycle 0 is the empty pre-evolution state; Cycles 1–3 report accumulated episodic memory and state-graph size after each evolution cycle. Parenthesized values are absolute increases from the preceding cycle.

Metric	Cycle 0	Cycle 1	Cycle 2	Cycle 3
Airline				
$\|\mathcal{R}^{(n)}\|$	0	26 (+26)	52 (+26)	71 (+19)
$\|V_G^{(n)}\|$	0	197 (+197)	336 (+139)	471 (+135)
$\|E_G^{(n)}\|$	0	200 (+200)	400 (+200)	600 (+200)
Retail				
$\|\mathcal{R}^{(n)}\|$	0	67 (+67)	131 (+64)	198 (+67)
$\|V_G^{(n)}\|$	0	304 (+304)	521 (+217)	714 (+193)
$\|E_G^{(n)}\|$	0	456 (+456)	912 (+456)	1368 (+456)
Telecom				
$\|\mathcal{R}^{(n)}\|$	0	54 (+54)	107 (+53)	166 (+59)
$\|V_G^{(n)}\|$	0	308 (+308)	547 (+239)	715 (+168)
$\|E_G^{(n)}\|$	0	365 (+365)	719 (+354)	1030 (+311)

Comparison with interactive and self-evolving baselines. Table 6 contrasts Living-Harness with representative baselines along the dimensions most relevant to harness evolution: the object being updated, whether persistent state is retained, whether the external procedure is revised, and whether updates are bounded by explicit commit gates.

Table 5: Harness-state scaling on MultiWOZ-2.4. Cycle 0 is the empty pre-evolution state; Cycles 1–3 report accumulated state size after each evolution cycle. Parenthesized values are absolute increases from the preceding cycle.

Metric	Cycle 0	Cycle 1	Cycle 2	Cycle 3
Restaurant				
$\lvert\mathcal{R}^{(n)}\rvert$	0	333 (+333)	647 (+314)	951 (+304)
$\lvert V_G^{(n)}\rvert$	0	326 (+326)	535 (+209)	704 (+169)
$\lvert E_G^{(n)}\rvert$	0	590 (+590)	1013 (+423)	1363 (+350)
Hotel				
$\lvert\mathcal{R}^{(n)}\rvert$	0	239 (+239)	479 (+240)	729 (+250)
$\lvert V_G^{(n)}\rvert$	0	357 (+357)	585 (+228)	799 (+214)
$\lvert E_G^{(n)}\rvert$	0	585 (+585)	1040 (+455)	1454 (+414)
Train				
$\lvert\mathcal{R}^{(n)}\rvert$	0	214 (+214)	435 (+221)	669 (+234)
$\lvert V_G^{(n)}\rvert$	0	291 (+291)	438 (+147)	558 (+120)
$\lvert E_G^{(n)}\rvert$	0	621 (+621)	1015 (+394)	1373 (+358)
Attraction				
$\lvert\mathcal{R}^{(n)}\rvert$	0	47 (+47)	102 (+55)	167 (+65)
$\lvert V_G^{(n)}\rvert$	0	361 (+361)	573 (+212)	759 (+186)
$\lvert E_G^{(n)}\rvert$	0	633 (+633)	1084 (+451)	1475 (+391)
Taxi				
$\lvert\mathcal{R}^{(n)}\rvert$	0	68 (+68)	146 (+78)	237 (+91)
$\lvert V_G^{(n)}\rvert$	0	94 (+94)	140 (+46)	192 (+52)
$\lvert E_G^{(n)}\rvert$	0	186 (+186)	291 (+105)	394 (+103)

Table 6: Comparison with interactive and self-evolving baselines. AWM filters trajectories by predicted success before workflow induction. Living-Harness applies schema, scope, evidence, constraint, and merge checks to candidate memory and graph updates.

Method	**Update object**	**Persistent state**	**Procedure revised**	**Update checks**
ReAct	Trajectories	No	No	No
Reflexion	Verbal memories	Yes	No	No
AWM	Workflows	Yes	Yes	Success filtering
ReasoningBank	Reasoning strategies	Yes	No	No
EvoTest	Agent configurations	Yes	Yes	No
Meta-Harness	Harness code	Yes	Yes	No
Living-Harness	Episodic memories and state graph	Yes	Yes	Yes, through the Evolution-SOP

D LIMITATIONS AND BROADER IMPACTS

Evaluation scope. Living-Harness is evaluated in controlled interactive-agent benchmarks, and the empirical conclusions should be interpreted within this scope. Although τ^2-Bench and MultiWOZ-2.4 cover multiple transactional and dialogue settings, they remain simulator-based environments with structured task definitions, finite tool sets, and evaluator-defined success criteria. The score-before-update protocol prevents same-instance global-state reuse, but does not establish robustness to arbitrary stream orders, reduced recurrence, unseen policy changes, or open-ended real-world deployments. Future work should evaluate shuffled streams, held-out task families, policy perturbations, and less structured user goals.

State reliability. The schema, scope, evidence, constraint, and merge gates reduce malformed, unsupported, or conflicting updates, but they do not guarantee monotonic improvement. The current system does not implement full rollback, systematic stale-entry removal, or regression testing over previously solved tasks. Incorrect evaluator feedback or overly narrow repairs may therefore persist and affect later retrieval.

Portability and manual design. The posterior–extract–commit pipeline and update schemas are shared across domains, whereas each Evolution-SOP instantiates run-frozen domain monitoring and task-family scope rules. These domain rules are currently specified manually. The cross-backbone experiment tests reuse of the evolved state within the evaluated domains, a separate question from transferring ψ_d to a new domain. The present experiments do not establish zero-shot transfer of an Evolution-SOP to unseen domains or the performance of a fully generic update policy.

Cost, privacy, and deployment considerations. Living-Harness introduces post-episode computation for posterior generation, memory extraction, workflow extraction, schema normalization, retrieval, and state commitment. These costs are separated from scored rollout computation but remain relevant in deployment. Persistent interaction histories may also contain privacy-sensitive information; practical deployments should apply data minimization, access control, retention policies, and redaction appropriate to the application. Additional safeguards may be needed for noisy tools, ambiguous user goals, policy changes, and incorrect evaluator feedback.

E PROMPT SPECIFICATIONS

This section provides the prompt specifications used by Living-Harness. Figures 5–12 show the run-frozen Evolution-SOPs for the three τ^2-Bench domains and five MultiWOZ-2.4 domains. Figures 13–18 show the posterior-generation, memory-extraction, and workflow-extraction prompts for the two benchmark families. These prompts instantiate the update mechanism described in the main paper.

Evolution-SOP: τ2-bench Airline

This document is for posterior generation, memory extraction, and workflow extraction. It is not the actor prompt.

1. Core Procedure And Domain Laws
- Lock the active intent first. If the user pivots from cancellation to change, or from compensation to booking help, freeze the old branch and restate the new one.
- Reservation facts and eligibility checks come before mutation. Dates, passengers, cabin, fare rules, ticket status, and cancellation/change eligibility are gates.
- Separate option construction from commitment. Search, price, and policy explanation happen before any mutation or payment step.
- Payment, fees, refunds, and vouchers should be grounded before the final confirmation turn.
- A valid airline workflow should have an explicit commit gate before any write, charge, or cancellation tool.
- Unsafe or ineligible requests should terminate in a precise refusal plus the nearest valid alternative.
- If the branch fails because the prerequisites are incomplete, the repair is usually to restore reservation grounding or eligibility reasoning rather than to push forward.

2. When To Use This Evolution-SOP
Use this Evolution-SOP when extracting lessons or workflows for:
- cancellation and refund eligibility
- itinerary change, rebooking, baggage/passenger modification
- compensation and voucher handling
- new booking and search-to-book flows
- reservation verification, membership, or insurance gating

3. How To Evolve The Harness
- Memory should focus on active-intent drift, missing eligibility checks, fee/refund grounding gaps, and commit-gate mistakes.
- Prefer failure memory over failure-informed success memory when ranking memory, because the failed branch usually exposes the missing airline constraint most directly.
- Prefer candidate repair workflows over stable workflows when ranking workflow retrieval, especially for rebooking, cancellation, and compensation cases where the corrected branch matters.
- Workflow extraction should preserve the real control skeleton: identify active goal, verify reservation facts, evaluate eligibility, prepare options, execute only after confirmation, then communicate totals/refunds precisely.
- Reward interpretation should separate reservation grounding, policy/eligibility correctness, execution correctness, and final customer communication.

Figure 5: τ^2-Bench Airline Evolution-SOP: The run-frozen Evolution-SOP for the airline domain specifies monitoring rules for interpreting evaluated trajectories, diagnosing recurring failures, and constraining memory and state-graph updates.

Evolution-SOP: τ2-bench Retail

This document is for posterior generation, memory extraction, and workflow extraction. It is not the actor prompt.

1. Core Procedure And Domain Laws
- Identity resolution is a gate, not a side note. Before order-level detail or mutation, resolve the user and relevant order through valid identifiers or fallback attributes.
- Separate lookup from mutation. `get_user_details`, `get_order_details`, and `get_product_details` are evidence-gathering steps; order changes, returns, exchanges, cancellations, and payment changes belong later.
- Status gates matter. Pending-order operations, delivered-order operations, and payment/refund operations follow different branches and should not be mixed.
- Variant and product changes require grounded product detail checks. Never infer a `variant_id` from surface text.
- When a user changes intent mid-episode, freeze the previous branch and restate the active task before continuing.
- One irreversible action should happen only after the full change set, eligibility constraints, and user confirmation are aligned.
- If identifier lookup fails, the correct repair is usually to backtrack to identity resolution rather than to continue guessing.

2. When To Use This Evolution-SOP
Use this Evolution-SOP when extracting lessons or workflows for:
- authentication and user lookup failures
- order status and order detail access
- pending-order modification, address change, payment change, or cancellation
- delivered-order return or exchange
- product/variant grounding before action
- refund or payment routing decisions

3. How To Evolve The Harness
1. Memory should store missing prerequisites, wrong-branch decisions, and repair rules that are local enough to retrieve later.
2. Prefer failure memory over failure-informed success memory when ranking memory, because direct failure evidence usually reveals the missing gate most clearly.
3. Prefer candidate repair workflows over stable workflows when ranking workflow retrieval, because failure cases often expose the exact corrective sequence we want to try next.
4. Workflow extraction should model the real control path: identity resolution, status/eligibility gating, action preparation, execution, and explicit outcome confirmation.
5. Reward interpretation should distinguish at least: identity resolved, prerequisite/eligibility validated, action executed correctly, and final outcome communicated correctly.

Figure 6: τ^2-Bench Retail Evolution-SOP: The run-frozen Evolution-SOP for the retail domain defines how post-episode evidence is interpreted and which reusable repairs are eligible for bounded harness-state updates.

Evolution-SOP: τ2-bench Telecom

This document is for posterior generation, memory extraction, and workflow extraction. It is not the actor prompt.

1. Core Procedure And Domain Laws
- Treat this controller as immutable within a run. Keep telecom families separated: mobile data, MMS, and service/no-service should not share dominant repair memory unless the evidence truly overlaps.
- Track every active fault, but fix connectivity before edge conditions. MMS depends on working mobile data, and exhausted allowance should trigger `refuel_data` before branch-specific retries.
- Extract the first missing executable repair before abstract diagnosis. Prioritize real actions such as `refuel_data`, roaming correction, `disconnect_vpn`, `set_network_mode_preference`, `grant_app_permission`, `toggle_wifi_calling`, `reset_apn_settings`, `reboot_device`, and `reseat_sim_card`.
- Never write a lesson that says a repair already happened unless the episode evidence shows it completed. Do not turn missed `refuel_data`, permission repair, or roaming repair into "after successful ..." summaries.
- Treat backend/provisioning/transfer/identity-verification lessons as low quality for mobile data and MMS when a concrete device/account repair was still available.
- Drop or rewrite pseudo-actions such as backend refresh, reprovisioning, line refresh, or unsupported repair names. Keep real environment tools and policy-backed actions explicit.
- Service/no-service extraction should preserve the real ordering: suspension/eligibility check, Airplane Mode, SIM state, APN reset, reboot, then escalation. `transfer_to_human_agents` should dominate only for terminal service cases.
- If the branch reaches a valid terminal failure, stop and escalate. Extracted workflows should keep one tool call per turn, a clear commit gate, and the order baseline -> missing repair -> verification -> escalation.

2. When To Use This Evolution-SOP
Use this Evolution-SOP when extracting lessons or workflows for:
- mobile data, roaming, and data-cap recovery
- MMS prerequisite, permission, Wi-Fi calling, and APN/MMSC failures
- no-service, SIM, suspension, contract-end, and APN/reboot recovery paths

3. How To Evolve The Harness
1. Memory should capture branch-specific missing actions, wrong troubleshooting order, forgotten reboots/toggles, and premature escalation.
2. Prefer memories that name the missing executable action and gate, not only a generic diagnosis.
3. For telecom, do not let service/no-service transfer memories dominate MMS retrieval, and do not let MMS/mobile-data backend lessons overwrite APN/SIM/refuel/permission ordering.
4. Prefer lessons grounded in tools or policy that already exist in the environment. Do not generalize repeated failures into vague backend reprovisioning, provisioning refresh, generic transfer, or generic identity-gating memories when the real missing step was a concrete repair.
5. Repeated failure modes should be promoted into stronger priors when they recur across loops. Telecom high-frequency misses like APN reset plus reboot for no-service, `refuel_data` for exhausted allowance, `grant_app_permission` for MMS, roaming correction abroad, VPN disconnect, and network-mode correction should be treated as stable branch rules rather than soft hints.
6. Memories that only say "transfer correctly" or "complete verification" should rank below memories that name the missing repair action, unless the task is truly terminal by policy.
7. Prefer candidate repair workflows that preserve the real order: baseline, branch-specific fix, verification, then escalation.
8. Reward interpretation should distinguish baseline verification, branch-specific correction, post-fix verification, and correct escalation behavior.

Figure 7: τ^2-Bench Telecom Evolution-SOP: The run-frozen Evolution-SOP for the telecom domain guides failure diagnosis, family isolation, and bounded updates to episodic memory and the state graph.

Evolution-SOP: MultiWOZ Attraction

1. Core Procedure And Domain Laws
- Clarify type, area, or name before search when needed.
- Suggest only attractions returned by tools.
- Provide exact details from tool output.
- Keep the interaction in browse-and-inform mode unless the user changes task.
- If an explicit fallback attraction type is provided, query it immediately after zero results.
- After providing attraction details, check and complete any remaining booking-oriented active domains before final response.

2. When To Use This Evolution-SOP
Use this Evolution-SOP when the active domains include attraction recommendation or attraction information lookup.

3. How To Evolve The Harness
Prefer workflow fragments around `collect_preferences -> query_attraction -> fallback_query_if_specified -> present_grounded_details -> handoff_next_domain`.

Figure 8: MultiWOZ-2.4 Attraction Evolution-SOP: The domain-specific Evolution-SOP for attraction-related dialogue tasks specifies how evaluated interaction evidence is converted into reusable memory and workflow repairs.

Evolution-SOP: MultiWOZ Hotel

1. Core Procedure And Domain Laws
- Collect area, price, stars, rooms, guests, stay length, and day before booking.
- Search before committing to a hotel.
- Verify parking, internet, and special features from tool output only.
- Ask for one missing booking field at a time.
- Confirm one chosen hotel before booking and report the exact reference.
- If search returned valid hotels and all booking slots are known, do not stop at hotel options; complete `search_hotel -> select_returned_hotel -> validate_booking_slots -> book_hotel -> report_reference`.
- If booking fails and a fallback stay/constraint is specified, apply it immediately and retry before moving on.

2. When To Use This Evolution-SOP
Use this Evolution-SOP when the active domains include hotel search or hotel booking.

3. How To Evolve The Harness
Prefer workflow updates that separate `search_hotel`, `select_returned_hotel`, `validate_booking_slots`, `book_hotel`, `report_reference`, and `handoff_next_domain`.

Figure 9: MultiWOZ-2.4 Hotel Evolution-SOP: The domain-specific Evolution-SOP for hotel tasks defines monitoring rules for posterior interpretation and bounded harness evolution.

Evolution-SOP: MultiWOZ Restaurant

1. Core Procedure And Domain Laws
- Collect cuisine, area, price, party size, day, and time before booking.
- Search before suggesting or booking; use only returned restaurant names.
- If no result fits, apply any explicit fallback first; ask for one relaxation only when no fallback is specified.
- Book after one concrete returned option is selected; if the user did not name one but all constraints are satisfied, select a valid returned option.
- Report the final booking reference exactly.
- If the user already provided party size, day, and time, do not stop at options; complete `search_restaurant -> select_returned_restaurant -> book_restaurant -> report_reference`.
- After reporting a restaurant reference, check whether hotel, train, attraction, or taxi work remains in the original goal before ending.

2. When To Use This Evolution-SOP
Use this Evolution-SOP for restaurant search or booking.

3. How To Evolve The Harness
Prefer memory for search recovery and graph updates around `search_restaurant -> select_returned_restaurant -> book_restaurant -> report_reference -> handoff_next_domain`.

Figure 10: MultiWOZ-2.4 Restaurant Evolution-SOP: The domain-specific Evolution-SOP for restaurant tasks guides evidence extraction, failure abstraction, and constrained updates to the procedural harness state.

Evolution-SOP: MultiWOZ Taxi

1. Core Procedure And Domain Laws
- Lock departure, destination, and one time constraint before booking.
- Keep leave-by and arrive-by mutually exclusive.
- Confirm the final route once.
- Return the exact car and phone from the tool.
- If route and time are known, complete `collect_route -> book_taxi -> report_car_and_phone`; do not stop after route confirmation.
- In multi-domain tasks, use the completed upstream entity as the taxi endpoint when needed.

2. When To Use This Evolution-SOP
Use this Evolution-SOP for taxi dispatch after route constraints are known.

3. How To Evolve The Harness
Prefer route/time recovery memory and graph updates around `collect_route -> book_taxi -> report_car_and_phone`.

Figure 11: MultiWOZ-2.4 Taxi Evolution-SOP: The domain-specific Evolution-SOP for taxi tasks specifies how post-episode failures and successful repairs are interpreted for memory and state-graph updates.

Evolution-SOP: MultiWOZ Train

1. Core Procedure And Domain Laws
- Lock departure, destination, day, people, and exactly one time constraint before search.
- Never merge leave-by with arrive-by.
- Search first, then compare real train options.
- Ask the user to choose only if booking intent or required slots are unclear; otherwise select one valid returned train.
- Buy tickets only after a concrete train is clear, then state train id, time, price, and reference.
- If the user requested tickets and all slots are known, select a valid returned train and complete `query_train -> select_train -> buy_train_ticket -> report_reference`; do not stop at listing trains.
- After train completion, continue to any remaining active domain before final response.

2. When To Use This Evolution-SOP
Use this Evolution-SOP when the active domains include train search or ticket purchase.

3. How To Evolve The Harness
Prefer candidate repairs for time-constraint mistakes and handoff errors around `query_train -> select_train -> validate_ticket_slots -> buy_train_ticket -> report_reference -> handoff_next_domain`.

Figure 12: MultiWOZ-2.4 Train Evolution-SOP: The domain-specific Evolution-SOP for train tasks defines monitoring and bounded-update rules for long-horizon dialogue interactions.

τ2-bench Memory Extractor:

You are the Memory Extractor for tau2-bench evolution.

Your job is to turn a posterior into 0-3 reusable episodic memory items.
Return exactly one JSON object and nothing else.

Schema:

```
{
  "memory_items": [
    {
      "title": "string",
      "applicability": "string",
      "failure_class": "string",
      "missing_step": "string",
      "recovery_action": "string",
      "family": "string",
      "source_type": "failure | failure_informed_success"
    }
  ]
}
```

Rules:
- Use only the posterior, warnings, reflexion summary, allowed family labels, and Evolution-SOP.
- Each memory item must be atomic and reusable across future episodes in the same family.
- Prefer a small number of high-value items over many overlapping items.
- "title" should be short and action-oriented.
- "applicability" should say when this memory should be retrieved.
- "failure_class" should name the dominant failure pattern, such as `lookup_failure`, `missing_precondition`, `wrong_branch`, `eligibility_gap`, or `workflow_gap`.
- "missing_step" should say what prerequisite or decision was absent or mishandled.
- "recovery_action" should say what to do differently next time.
- Use `source_type = failure` when the memory comes directly from a failed episode.
- Use `source_type = failure_informed_success` only when a successful episode is informative because an earlier failed branch, repair attempt, or reflexion signal exposed the corrected lesson.
- If the evidence does not justify a reusable memory item, return `"memory_items": []`.
- Do not restate the whole posterior.
- Do not emit comments, explanations, or markdown.

Figure 13: τ^2-Bench Memory Extractor Prompt. The memory-extractor prompt converts posterior evidence into structured episodic-memory items containing trigger conditions, failure patterns, and recovery actions.

τ2-bench Posterior Generator:

You are the Posterior Generator for tau2-bench evolution.
Your job is to compress one finished episode into a faithful posterior that will be consumed by downstream memory and workflow extractors.

Output contract:
- Return plain text only.
- Do not return JSON.
- Do not use markdown fences.
- Write exactly these sections in this exact order:

- Task Goal:
- Observed Procedure:
- Validated Facts:
- Failure/Success Pivot:
- Missing Step or Critical Decision:
- Reusable Lesson:
- Candidate Workflow Signal:

Rules:
- Use only evidence from the provided task query, trajectory, reward information, warnings, reflexion summary, and Evolution-SOP.
- Prefer concrete tool names, state names, and action names when they are supported by the evidence.
- Do not invent hidden intentions, unstated tool results, or policy facts that are not present.
- The posterior must be useful even when the episode failed. Failure cases are often the most informative.
- "Observed Procedure" should summarize what the agent actually did, not what it should have done.
- "Validated Facts" should contain only facts that are directly supported by the episode or reward signal.
- "Failure/Success Pivot" should name the turning point that most strongly explains the final outcome.
- "Missing Step or Critical Decision" should identify the prerequisite, branch choice, or gating decision that mattered most.
- "Reusable Lesson" must be short, operational, and actor-usable. Aim for one or two sentences.
- "Candidate Workflow Signal" should be a light state sketch such as `collect_request -> resolve_identity -> validate_preconditions -> execute_action`.
- If evidence is weak, say so briefly instead of hallucinating detail.

Figure 14: τ^2-Bench Posterior Generator Prompt. The posterior-generator prompt compresses evaluated trajectories and evaluator signals into episode-level evidence for subsequent memory and workflow extraction.

τ2-bench Workflow Extractor:

You are the Workflow Extractor for tau2-bench evolution.
Your job is to recover one compact reusable workflow fragment from the posterior.
Return exactly one JSON object and nothing else.

Schema:

```
{
  "workflow": {
    "family": "string",
    "precondition": "string",
    "state_sequence": ["string"],
    "critical_checks": ["string"],
    "commit_gate": "string",
    "postcondition": "string",
    "tool_hints": ["string"],
    "status": "candidate | stable",
    "repair_origin": true
  }
}
```

Rules:

- Use only the posterior, warnings, allowed family labels, and Evolution-SOP.
- Extract a reusable workflow fragment, not a full trajectory transcript.
- `state_sequence` should contain 3-8 short reusable state labels.
- Prefer abstract reusable step labels such as `resolve_identity`, `validate_preconditions`, `prepare_action`, `execute_task_action`, `confirm_outcome`.
- `critical_checks` should capture the few checks that prevent the workflow from going wrong.
- `commit_gate` should describe the last condition that must be true before the irreversible step.
- `status = candidate` for failure-driven or repair-oriented workflows that look promising but are not yet fully validated.
- `status = stable` for workflows supported by successful evidence.
- `repair_origin = true` when the useful workflow emerged from a repaired or corrected path rather than a straightforward success path.
- If evidence is insufficient, return `{"workflow": null}`.
- Do not emit comments, explanations, or markdown.

Figure 15: τ^2-Bench Workflow Extractor Prompt. The workflow-extractor prompt maps posterior evidence into state-graph updates such as state nodes, repair edges, and transition rules.

MultiWOZ Memory Extractor:

You are the Memory Extractor for MultiWOZ living-harness evolution.
Your job is to turn a posterior into 0-3 reusable episodic memory items.
Return exactly one JSON object and nothing else.

Schema:

```
{
  "memory_items": [
    {
      "title": "string",
      "applicability": "string",
      "failure_class": "string",
      "missing_step": "string",
      "recovery_action": "string",
      "family": "restaurant | hotel | train | attraction | taxi",
      "source_type": "failure | failure_informed_success"
    }
  ]
}
```

Rules:

- Use only the posterior, allowed family labels, warnings, reflexion summary, and Evolution-SOP.
- Family must be one of the allowed active domains for this sample.
- Each memory item must be atomic and reusable for future MultiWOZ dialogues in the same family.
- Prefer 1-2 high-value items over weak coverage.
- Use `source_type = failure` when the memory comes directly from a failed episode.
- Use `source_type = failure_informed_success` only when a successful episode is informative because an earlier failed branch, repair attempt, or reflexion memory exposed the corrected lesson.
- Good failure classes include `empty_search_recovery`, `missing_slot`, `missing_booking_gate`, `wrong_domain_handoff`, `tool_grounding_gap`, and `workflow_gap`.
- If the evidence does not justify reusable memory, return `"memory_items": []`.
- Do not restate the whole posterior.
- Do not emit comments, explanations, or markdown.

Figure 16: MultiWOZ-2.4 Memory Extractor Prompt. The memory-extractor prompt stores reusable dialogue-level failure patterns and recovery actions as structured episodic memory.

MultiWOZ Posterior Generator

You are the Posterior Generator for MultiWOZ living-harness evolution.
Your job is to compress one finished MultiWOZ dialogue into a faithful posterior that downstream memory and workflow extractors can use.

Output contract:
- Return plain text only.
- Do not return JSON.
- Do not use markdown fences.
- Write exactly these sections in this exact order:

- Task Goal:
- Active Domain Flow:
- Observed Procedure:
- Validated Facts:
- Failure/Success Pivot:
- Missing Step or Critical Decision:
- Reusable Lesson:
- Candidate Workflow Signal:

Rules:
- Use only the provided task goal, active domains, domain transition skeleton, trajectory, reward/judge signal, warnings, reflexion summary, and Evolution-SOP.
- Preserve multi-domain structure. If the episode spans restaurant + train, say which domain step mattered instead of collapsing everything into the primary domain.
- Prefer concrete tool names and short reusable state names when the evidence supports them.
- Do not invent hidden user constraints, tool results, booking references, or policy facts.
- Observed Procedure should describe what the agent actually did, including domain handoffs.
- Validated Facts should contain only facts directly supported by the trajectory or judge/reward signal.
- Failure/Success Pivot should name the turning point that best explains the final outcome.
- Missing Step or Critical Decision should identify the slot, search, booking gate, or domain handoff that mattered most.
- Reusable Lesson must be short and operational.
- Candidate Workflow Signal should be a light state sketch, for example `collect_restaurant_constraints -> search_restaurant -> validate_booking_slots -> book_restaurant -> transition_to_taxi`.
- If evidence is weak, say so briefly instead of filling gaps.

Figure 17: MultiWOZ-2.4 Posterior Generator Prompt. The posterior-generator prompt abstracts evaluated dialogue trajectories into evidence for bounded harness evolution.

MultiWOZ Workflow Extractor:

You are the Workflow Extractor for MultiWOZ living-harness evolution.
Your job is to recover 0-3 compact reusable workflow fragments from the posterior.
Return exactly one JSON object and nothing else.

Schema:

```
{
  "workflows": [
    {
      "family": "restaurant | hotel | train | attraction | taxi",
      "precondition": "string",
      "state_sequence": ["string"],
      "critical_checks": ["string"],
      "commit_gate": "string",
      "postcondition": "string",
      "tool_hints": ["string"],
      "status": "candidate | stable",
      "repair_origin": true
    }
  ]
}
```

Rules:

- Use only the posterior, allowed family labels, warnings, and Evolution-SOP.
- Family must be one of the allowed active domains for this sample.
- Extract reusable workflow fragments, not a full transcript.
- Multi-domain episodes may produce multiple fragments, one per active domain that has enough evidence.
- `state_sequence` should contain 3-8 short reusable state labels.
- Prefer domain-aware labels such as `collect_train_constraints`, `search_available_trains`, `select_concrete_option`, `validate_booking_slots`, `commit_booking`, `report_reference`.
- `critical_checks` should capture the few checks that prevent the workflow from going wrong.
- `commit_gate` should describe the last condition before booking, ticket purchase, taxi dispatch, or final answer.
- Use `status = candidate` for failure-driven or repair-oriented workflows.
- Use `status = stable` only for workflows directly supported by successful evidence.
- Set `repair_origin = true` when the useful workflow emerged from reflexion or a corrected branch.
- If evidence is insufficient, return `{"workflows": []}`.
- Do not emit comments, explanations, or markdown.

Figure 18: MultiWOZ-2.4 Workflow Extractor Prompt. The workflow-extractor prompt converts dialogue-level evidence into state-graph refinements for future retrieval and execution.