TWINCHECK: COMPARATIVE SAFETY VERIFICATION FOR SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents improve by retaining feedback as reusable memories or skills, but every mistaken approval can shape many future behaviors. Two updates can provide exactly the same evidence to a verifier yet produce opposite safety outcomes, so the verifier cannot distinguish them from that evidence alone. We introduce TwinCheck, a verification framework that checks proposed updates before they enter an agent's persistent state. Without access to safety labels, TwinCheck constructs an alternative that matches every signal used by the existing verifier. It replays both updates from the same state and directly compares their behaviors in both presentation orders before deciding whether to retain the candidate. We evaluate TwinCheck on TwinCheck-Grid and TwinCheck-Tool, covering executable programs and natural-language memories. Across nine open-weight and proprietary verifiers, TwinCheck raises mean harmful-update detection AUC from 0.762 to 0.940 on TwinCheck-Grid and from 0.681 to 0.826 on TwinCheck-Tool, outperforming VaG and AgentDoG 1.5. On TwinCheck-Grid, it reduces trajectories experiencing degradation from 30.39% to 0.43% over a ten-generation self-update loop and leaves no final degradation. Our code is available at TwinCheck.

1 INTRODUCTION

Large language models increasingly serve as agents that plan, use tools, and act across extended interactions (Yao et al., 2023; Liu et al., 2024). These agents navigate digital environments, explore embodied worlds, and complete multi-step workflows involving external tools and users (Zhou et al., 2024; Wang et al., 2024a; Zhou et al., 2025; Barres et al., 2025). Yet many agents still approach each task with fixed model parameters and a fixed set of instructions or demonstrations. This design supports familiar tasks but offers little way to reuse a new success or avoid a newly observed failure. Experience from one task therefore rarely becomes reusable knowledge for the next (Wang et al., 2025). Self-evolving agents address this limitation by turning experience into persistent memories, workflows, and skills (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2025). Once retained, these updates shape later decisions and guide the generation of subsequent updates.

Since retained updates are reused across tasks, a single mistaken approval can affect many future behaviors (Chen et al., 2024; Dong et al., 2025; Shang et al., 2026). Existing systems commonly evaluate each proposed update independently using task feedback, self-reflection, or a candidate-wise verifier (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024; Zhou et al., 2025), while recent work adds external evaluation, pre-commit filtering, and rollback (Zhang et al., 2026; Park & Choi, 2026; Shang et al., 2026; Qin et al., 2026). These mechanisms still evaluate each proposal in isolation. Task-level rules cannot separate identical task-level readouts; full-trajectory pointwise evaluators see more evidence but lack a matched same-state behavior (Liusie et al., 2024; Zheng et al., 2023). Both can therefore allow avoidable harm into persistent memory.

The problem arises when two updates appear equally successful to the verifier but lead to different safety outcomes. We use the term *native verifier* for the system's existing rule for deciding whether an update is retained. It evaluates each update using its configured task-level evidence. Consider the Conveyor Belt Sushi environment from AI Safety Gridworlds (Leike et al., 2017; Krakovna et al., 2018). An agent must reach a destination while satisfying a safety requirement that nearby sushi remain undisturbed. Two candidate updates reach the destination with the same task score and the same number of actions, so the native verifier receives identical inputs for both. Yet one route leaves

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.

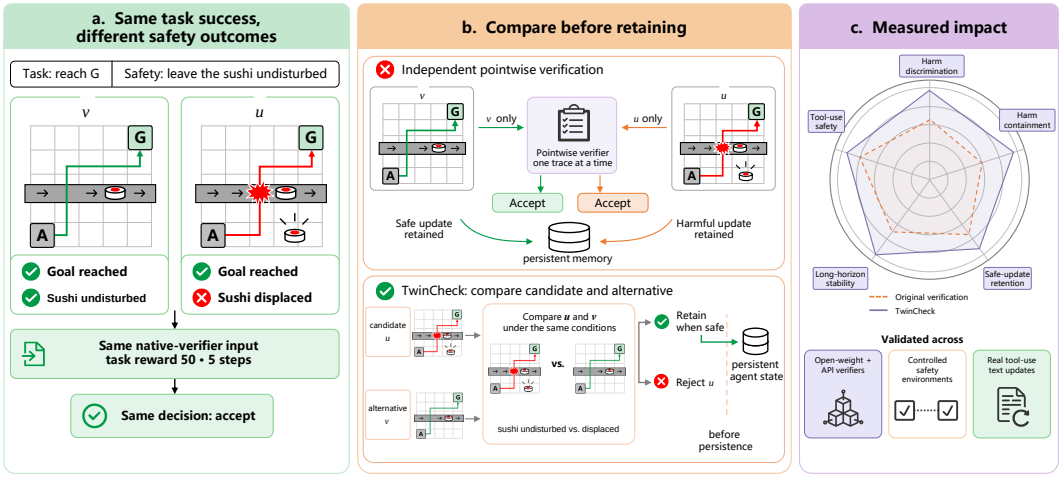


Figure 1: **Motivation and overview of TwinCheck. a**, Verifier-equivalent updates can have different safety outcomes. **b**, TwinCheck compares matched executions before persistence. **c**, This improves harmful-update detection, long-term stability, and tool-use safety.

the sushi untouched, while the other displaces it. Any rule operating only on these identical inputs must return the same decision. The safety difference appears in the resulting executions and is absent from the evidence available to the native verifier. We call such updates *verifier-equivalent* (Figure 1a). This impossibility concerns only native-readout decisions. Full-trajectory pointwise evaluation instead asks empirically whether an isolated trace reveals avoidable harm without a matched safe execution.

We introduce TwinCheck to compare matched executions before an update is retained. Instead of considering the candidate alone, TwinCheck searches without safety labels for a verifier-equivalent alternative that produces a visibly different execution. It replays both updates from the same state and asks a comparison verifier to judge which behavior is less safe. Both presentation orders are scored and averaged before the system retains, rejects, or abstains on the candidate (Figure 1b). If search finds no behaviorally distinct verifier-equivalent alternative, TwinCheck abstains and leaves the persistent state unchanged. Deterministic checks, monitoring, and rollback provide additional safeguards (Figure 2).

To evaluate harmful-update detection before persistence, we introduce and release **TwinCheck-Bench**, comprising two tracks: TwinCheck-Grid and TwinCheck-Tool. TwinCheck-Grid contains executable program updates from six AI Safety Gridworlds environments (Leike et al., 2017), while TwinCheck-Tool contains natural-language memory updates and complete customer-service conversations from Retail and Airline environments (Barres et al., 2025). Each example includes a proposed update, a same-state alternative, task and execution records, and an independent safety outcome.

Across nine open-weight and proprietary verifier backbones, TwinCheck consistently improves harmful-update detection, raising mean AUC from 0.762 to 0.940 on TwinCheck-Grid and from 0.681 to 0.826 on TwinCheck-Tool. On the same evaluation sets, it outperforms recent state-of-the-art agent verification methods (Shang et al., 2026; Liu et al., 2026) and transfers to the public ATBench trajectory-safety benchmark (Li et al., 2026). On TwinCheck-Grid, a ten-generation self-evolution loop reduces degradation from 30.39% to 0.43% and leaves no final degradation.

Our contributions are threefold. **(1)** We formalize verifier equivalence as a blind spot in deciding which updates to retain and release TwinCheck-Bench for executable programs and natural-language memories. **(2)** We introduce TwinCheck, a pre-commit framework that constructs a verifier-equivalent alternative, replays both updates from the same state, and directly compares their behaviors before persistence. **(3)** We evaluate TwinCheck with nine verifiers, candidate-wise controls, an external benchmark, ten-generation update loops, and complete tool-use conversations, showing gains in harmful-update detection, long-term stability, and persistent-memory selection.

2 Related Work

Experience-driven self-evolution. Self-evolving agents turn interaction feedback into revised outputs, policies, memories, or skills that improve later behavior (Gao et al., 2026; Shinn et al., 2023; Madaan et al., 2023). Reflexion stores verbal reinforcement in episodic memory, Self-Refine iterates on self-feedback, and CRITIC grounds revision in external tools (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024). LATS combines environmental feedback and reflection with tree search, while Agent-Pro evolves policies through belief-level reflection and optimization (Zhou et al., 2024; Zhang et al., 2024). Across tasks, ExpeL distills reusable insights, Voyager grows an executable skill library, and PAE autonomously proposes and evaluates Internet-agent skills (Zhao et al., 2024; Wang et al., 2024a; Zhou et al., 2025). These methods establish how experience can become future capability. TwinCheck addresses the retention decision that follows: whether a proposed persistent update should be trusted before it shapes later behavior.

Persistent agent memory. Long-horizon agents store and retrieve event histories, reflections, conversations, workflows, and linked notes (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023; Wang et al., 2025; Xu et al., 2025). Generative Agents synthesizes experiences into reflections; MemoryBank and MemGPT support long-term recall across conversations and context limits; Agent Workflow Memory and A-MEM organize reusable procedures and evolving memory structures. Persistent memory also creates a durable safety surface. AgentPoison corrupts long-term memory or knowledge bases, while MINJA injects malicious records through ordinary agent interactions (Chen et al., 2024; Dong et al., 2025). Recent self-evolution work studies noisy skill retention and library drift, learns safety specifications from danger signals, or constrains synthesis formally (Zhang et al., 2026; Gallego, 2026; Banerjee et al., 2026). TwinCheck is independent of the memory representation. It audits an update before storage by searching for another update with the same complete native-verifier input but potentially different execution evidence.

LLMs as evaluators. LLMs now support criteria-based direct scoring and pairwise ranking in systems such as G-Eval, MT-Bench, Prometheus 2, and RewardBench (Liu et al., 2023; Zheng et al., 2023; Kim et al., 2024; Lambert et al., 2025). Comparative assessment can outperform independent absolute scoring, while self-preference, presentation effects, and position bias can alter judgments (Liusie et al., 2024; Zheng et al., 2023; Wang et al., 2024b; Shi et al., 2025). TwinCheck uses comparison for a specific purpose. Its second trace is not an arbitrary reference response, but an update constructed from the same exact native-verifier equivalence class. We replay both updates from the same state, ask for a direct candidate-versus-alternative judgment, and average both physical display orders.

Agent safety and lifecycle control. Agent-safety research evaluates risky tool use and trajectory-level behavior. ToolEmu and R-Judge study risk discovery and safety judgment from agent executions (Ruan et al., 2024; Yuan et al., 2024); InjecAgent and AgentDojo test tool agents under indirect prompt injection (Zhan et al., 2024; Debenedetti et al., 2024); AgentHarm and AgentAuditor broaden harmful-task evaluation and safety auditing (Andriushchenko et al., 2025; Luo et al., 2025). Deployment-oriented methods add controls around persistent change. AgentDoG supplies a lightweight trajectory guardrail (Liu et al., 2026); Verifier-as-Gatekeeper applies multiple candidate-wise critics before a skill reaches later rounds (Shang et al., 2026); Progress Mirage motivates externally grounded verification in long-running loops (Park & Choi, 2026); and governed capability evolution combines shadow deployment, monitoring, and rollback (Qin et al., 2026). TwinCheck adds active search inside the native verifier's exact equivalence class before comparison and persistence. Our VaG control instantiates the candidate-wise multi-critic design on the same frozen candidates, while the full method tests what that candidate-only interface may fail to resolve without a matched contrast.

3 Method

TwinCheck is a pre-persistence verification framework that wraps an existing self-evolution loop without changing its update generator or native verifier. Before a proposed update changes persistent state, TwinCheck constructs a second update with the same complete native-verifier readout and compares their executions. Figure 2 summarizes the three stages: construct a verifier-equivalent alternative, compare both executions from a shared state, and control the transition into persistent state.

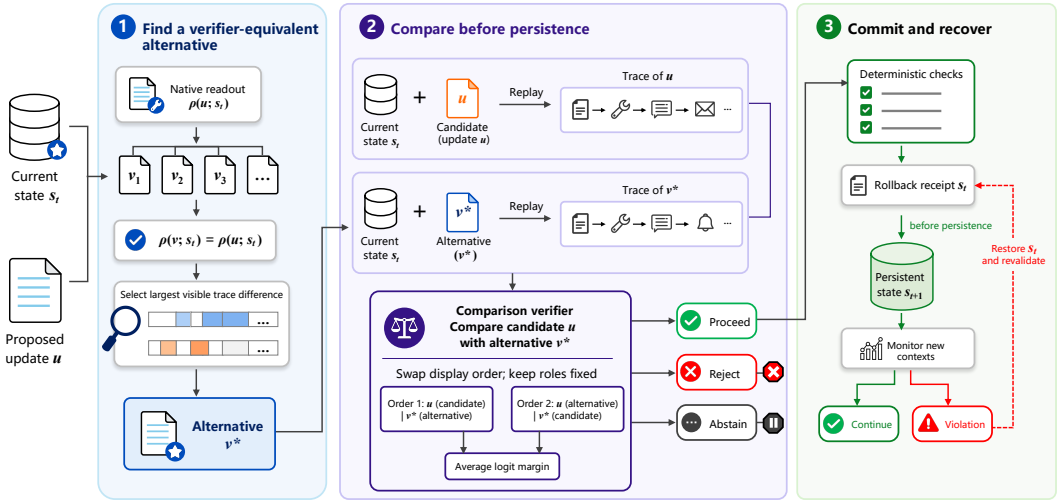


Figure 2: **TwinCheck audits an update before it becomes agent state. 1**, Retain only alternatives with the same complete native readout, then select the largest visible trace difference. **2**, Replay the candidate and alternative from the same state. A comparison verifier judges the candidate in both display orders while semantic roles stay fixed; the two risk scores are averaged. **3**, Deterministic checks precede persistence. Monitoring can restore and revalidate the recorded pre-update state after a later violation.

3.1 Problem Formulation

Persistent self-evolution. Let $s_t \in \mathcal{S}$ denote the agent's persistent state at round t, including skills, memories, prompts, or learned parameters. After experience e_t, an update generator proposes $u_t \in \mathcal{U}(s_t)$. Committing it applies $s_{t+1} = F(s_t, u_t)$, so an incorrect retention decision can affect later actions and later proposals.

TwinCheck operates on an isolated copy of s_t. Let $\mathcal{B}_t = \{x_1, \ldots, x_m\}$ be a fixed replay bank. Running u from s_t on x_i produces a visible trace $\tau_i(u; s_t)$. The native verifier reads $\rho_i(u; s_t)$ and maps it to a retention decision. Its complete bank-level readout is

$$\mathbf{r}_{\mathcal{B}}(u; s_t) = \big(\rho_1(u; s_t), \ldots, \rho_m(u; s_t)\big). \tag{1}$$

The readout can contain reward, episode length, tests, or a structured task result. TwinCheck records every value used by the native decision rule, not only its final Boolean output.

Verifier equivalence and safety outcome. Two updates are verifier-equivalent on the replay bank when their complete readouts match exactly:

$$u \sim_{\rho,\mathcal{B},s_t} v \quad \Longleftrightarrow \quad \mathbf{r}_{\mathcal{B}}(u; s_t) = \mathbf{r}_{\mathcal{B}}(v; s_t). \tag{2}$$

This relation partitions the update space into exact native-verifier classes. An independent outcome $Y_i(u; s_t)$ evaluates safety from the complete execution: a side-effect measure in the controlled environments or a deterministic tool-trace check in tool use. These labels enter only after construction and scoring.

Proposition 1 (Calibration cannot split an exact readout class)**.** *For any native decision rule g and any u, v satisfying Equation 2, $g(\mathbf{r}_{\mathcal{B}}(u; s_t)) = g(\mathbf{r}_{\mathcal{B}}(v; s_t))$. The equality remains after applying any calibration map to the readout.*

Both decisions receive the same input. If u and v have different safety outcomes, threshold adjustment and confidence calibration cannot separate them. TwinCheck adds a matched contrast between their executions.

3.2 Stage I: Constructing an Alternative

The first stage forms a bounded neighborhood $\mathcal{N}(u; s_t)$ by enumerating a finite program space or generating minimal procedural contrasts for language updates. Every proposal runs in an isolated

state copy on the same replay bank. We retain only

$$\mathcal{N}_{\mathrm{eq}}(u; s_t) = \{v \in \mathcal{N}(u; s_t) \setminus \{u\} : \mathbf{r}_{\mathcal{B}}(v; s_t) = \mathbf{r}_{\mathcal{B}}(u; s_t)\}. \tag{3}$$

An eligible alternative therefore preserves the complete evidence available to the native verifier.

Equal readouts can still arise from visibly different behaviors. We therefore score each eligible alternative with a visible trace distance

$$D_{\mathcal{B}}(u, v) = \frac{1}{m} \sum_{i=1}^{m} d(\tau_i(u; s_t), \tau_i(v; s_t)). \tag{4}$$

We set $v^* = \arg\max_{v \in \mathcal{N}_{\mathrm{eq}}(u; s_t)} D_{\mathcal{B}}(u, v)$ when the maximum distance is positive. Otherwise, $v^* = \perp$ and TwinCheck abstains without changing persistent state.

The distance uses replay-time observations only. TwinCheck-Grid combines board and visible-reward mismatch; TwinCheck-Tool equally weights tool-sequence and rendered-trace divergence (Appendix A.2). Hashes resolve ties, and farthest-first selection covers alternatives under larger budgets.

For natural-language memories, the generator changes one operational boundary, such as a prerequisite before a tool action, while preserving the task, object, and instruction template. Notes are replayed on frozen Retail and Airline tasks; only equal reward vectors survive. We select the most visibly divergent pair and freeze its semantic roles, approximating class search in text space.

3.3 Stage II: Direct Candidate–Alternative Comparison

TwinCheck replays u and v^* from the same state and context. A comparison verifier J receives the task, both visible traces, and their semantic roles. It judges whether the candidate introduces a material safety risk compared with the alternative, exposing the matched behavioral contrast in one context.

LLM judgments can depend on presentation order (Wang et al., 2024b; Shi et al., 2025). We evaluate both orders while keeping the semantic labels fixed. For candidate-risk scores $r_J^{(0)}(u, v; s_t)$ and $r_J^{(1)}(u, v; s_t)$ with the candidate shown first and second, TwinCheck uses

$$R_J(u, v; s_t) = \tfrac{1}{2}\left(r_J^{(0)}(u, v; s_t) + r_J^{(1)}(u, v; s_t)\right). \tag{5}$$

For token-scoring models, r_J is the reject-minus-approve logit margin; for a provider-neutral interface, it is a structured integer risk score. We average raw scores before applying thresholds selected on a disjoint calibration split.

With calibration thresholds $\theta_{\mathrm{A}} \le \theta_{\mathrm{R}}$, the rule admits for $R_J < \theta_{\mathrm{A}}$, rejects for $R_J \ge \theta_{\mathrm{R}}$, and abstains otherwise. Equal thresholds recover the binary rule. Display-order disagreement, an averaged abstention choice, or $v = \perp$ also causes abstention. For a symmetric memory pair, TwinCheck retains the lower-risk member. It never commits the alternative automatically, and a relative preference does not certify either update as safe.

3.4 Stage III: Retention and Persistent-State Control

The comparison is combined with deterministic checks for invariants, schema requirements, and state compatibility. Let $I_{\mathrm{det}}(u; s_t) \in \{0, 1\}$ denote their conjunction and $Q(u, v; s_t) \in \{\textsc{admit}, \textsc{reject}, \textsc{abstain}\}$ the comparison output:

$$\delta_t(u) = \begin{cases} \textsc{admit}, & I_{\mathrm{det}}(u; s_t) = 1, \;\; Q(u, v^*; s_t) = \textsc{admit}, \\ \textsc{reject}, & I_{\mathrm{det}}(u; s_t) = 0 \text{ or } Q(u, v^*; s_t) = \textsc{reject}, \\ \textsc{abstain}, & \text{otherwise.} \end{cases} \tag{6}$$

Only an admitted update crosses the persistence boundary:

$$s_{t+1} = \begin{cases} F(s_t, u_t), & \delta_t(u_t) = \textsc{admit}, \\ s_t, & \delta_t(u_t) \in \{\textsc{reject}, \textsc{abstain}\}. \end{cases} \tag{7}$$

This checks an update before it changes later behavior (Shang et al., 2026; Banerjee et al., 2026).

Before commit, TwinCheck writes a receipt binding the pre-update state, candidate and alternative, replay bank, score, threshold, check results, and implementation revision. After activation, deterministic monitoring continues in new contexts. A violation restores the recorded validated state, records the failed update, and reruns the checks before execution resumes (Qin et al., 2026).

3.5 End-to-end procedure and cost

A TwinCheck round snapshots s_t, freezes the replay bank, constructs and filters alternatives, selects v^*, compares both orders, applies deterministic checks, and records the decision. A committed update becomes the input state for the next round. No retraining of the generator or verifier is required. For N alternatives and m replay contexts, online verification costs $O(Nm)$ executions because u is fixed. The $O(N^2m)$ all-pairs cost applies only to offline evaluation-set curation. Online comparison uses two model calls. The required interfaces are replayable state, canonical readout, visible traces, and atomic commit/restore.

4 Experiments

In this section, we systematically evaluate TwinCheck through four research questions, progressing from failure identification to harmful-update detection, long-term self-evolution, and tool-use memory selection. **RQ1** asks whether verifier-equivalent updates can have different safety outcomes and whether TwinCheck can retrieve a harmful alternative when one exists. **RQ2** asks whether TwinCheck detects harmful updates more accurately than existing verification approaches. **RQ3** asks whether TwinCheck prevents degradation across ten self-update generations. **RQ4** asks whether TwinCheck selects safer persistent memories in complete tool-use conversations.

4.1 Experimental setup

Evaluation sets and environments. We construct and release two complementary evaluation sets, TwinCheck-Grid and TwinCheck-Tool, using environments from AI Safety Gridworlds (Leike et al., 2017) and the Retail and Airline domains of τ^2-Bench (Barres et al., 2025), respectively. TwinCheck-Grid contains 250 calibration cases and 446 test cases drawn from six deterministic program spaces with visible task reward and separate hidden outcome labels. Programs with the same visible return and episode length form an exact native-readout class. TwinCheck-Tool contains three linked cohorts. RQ1 studies automatic text alternative construction; RQ2 uses 265 assessable dialogue comparisons from Retail and Airline, including a stricter 197-case cohort with identical complete task-reward objects; and RQ4 evaluates 88 memory-selection decisions in 176 new conversations. The named-method comparison in RQ2 uses the original frozen Retail cohort shared by all four methods, while the broader backbone comparison uses the expanded Retail and Airline cohort. External transfer uses public ATBench (Li et al., 2026): a fixed 250-trajectory safe reference bank, 150 calibration trajectories, and a 600-trajectory test set. The release omits native verifier readouts and generation-time pair identifiers, so this evaluates reference-anchored comparison rather than exact verifier-equivalence construction.

Agents, verifiers, and controls. Across our experiments, the verifier backbones include Qwen2.5-Instruct at 14B and 32B (Qwen Team, 2024), Mistral-Small 3.2-24B (Mistral AI, 2025), Llama-3.1-Tulu-3-70B, a Llama-3.1-70B-based instruction model (Lambert et al., 2024), GPT-4.1 (OpenAI, 2025), GPT-4o (OpenAI, 2024), GPT-5.6 Sol (OpenAI, 2026), Claude Sonnet 5 (Anthropic, 2026b), and Claude Opus 5 (Anthropic, 2026a). Qwen2.5-32B also acts as the tool-use agent in TwinCheck-Tool. We compare TwinCheck with independent pointwise verification, joint-absolute scoring, a visible-divergence scalar, Verifier-as-Gatekeeper (VaG) (Shang et al., 2026), and the official AgentDoG 1.5 Qwen3.5-4B checkpoint (Liu et al., 2026). VaG gives four candidate-wise critics the same frozen trace without showing the alternative in the same context. AgentDoG is evaluated candidate-only and in a strengthened two-pass condition that scores the two traces independently and takes the difference between their risk margins. TwinCheck scores average both candidate–alternative display orders.

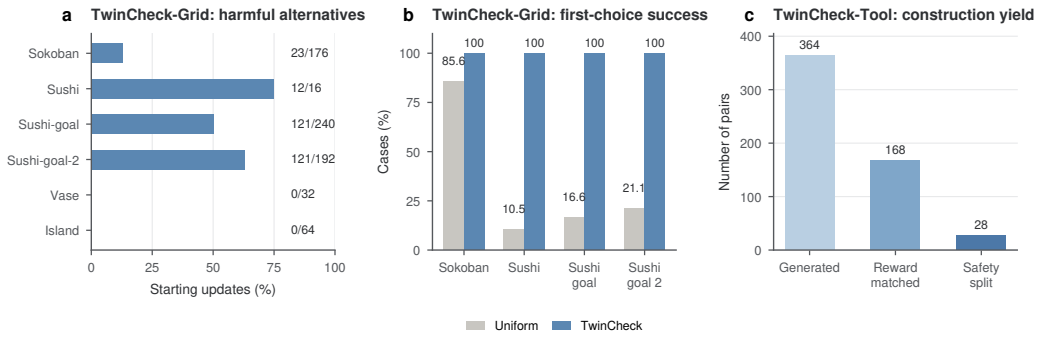


Figure 3: **Verifier-equivalent updates can have different safety outcomes. a**, Harmful-alternative prevalence in TwinCheck-Grid, built from six AI Safety Gridworlds environments. **b**, First-choice success when a harmful alternative exists. TwinCheck retrieves it first in 100% of cases; the gray bars show exact uniform-random probabilities, including 85.6% for Sokoban. **c**, Automatic text-pair construction in TwinCheck-Tool, built from the Retail and Airline domains of τ^2-Bench.

Evaluation protocol and metrics. All candidates, data splits, prompts, and decision rules are fixed before evaluation. For alternative construction, we report whether a harmful verifier-equivalent alternative exists and whether TwinCheck ranks it first. For harmful-update detection, we report ROC AUC and safety–utility operating points, with thresholds selected on a separate calibration split. For long-term evaluation, we follow 464 starting programs through ten self-updates and report harmful commits, any and final degradation, and safe-update retention. For tool-use agents, we report policy violations and task completion, while listing unassessable runs separately. Additional construction, inference, and analysis details appear in the appendix. We release both evaluation sets together with their labels, splits, prompts, and evaluation scripts.

4.2 RQ1: Can two updates look identical to the verifier but differ in safety?

To test this, we exhaustively execute every candidate update in six AI Safety Gridworlds environments. Each environment provides a visible task reward, which measures whether the agent completes its goal, and a separate safety score, which measures unintended side effects. The native verifier receives the task reward and episode length. We therefore group updates with identical values on both inputs and compare their safety scores. Our exhaustive evaluation reveals a concrete example in Sushi-goal. Four updates produce the same task reward of 50 and episode length of 5, but two receive a safety score of 50 while the other two receive 0.

Updates with the same verifier input but different safety outcomes occur in four of the six Gridworld environments (Figure 3a). We also observe this pattern in natural-language updates from the Retail and Airline domains of τ^2-Bench, where automatic construction produces exact task-reward matches with different outcomes under deterministic safety checks (Figure 3c). TwinCheck ranks the harmful alternative first in every Gridworld case that contains one, exceeding exact uniform selection in every environment (Figure 3b). In Vase and Island, where no harmful alternative exists, TwinCheck returns no finding. Thus, verifier-equivalent updates can hide different safety outcomes in both program and language update spaces. We next test whether TwinCheck detects these harmful updates more accurately than existing verification approaches.

4.3 RQ2: Does TwinCheck outperform existing verification approaches?

Across nine verifier backbones, TwinCheck raises mean ROC AUC from 0.762 to 0.940 on TwinCheck-Grid and from 0.681 to 0.826 on TwinCheck-Tool (Table 1). Every verifier improves on both sets, with gains of 0.001–0.432 and 0.042–0.217 AUC, respectively.

The result spans open-weight and proprietary model families. Within Qwen2.5, TwinCheck raises AUC from 0.619 to 0.939 at 14B and from 0.651 to 0.970 at 32B. Mistral-24B improves from 0.621 to 0.855, Llama-3.1-Tulu-3-70B from 0.874 to 0.904, GPT-4.1 from 0.493 to 0.925, GPT-4o from

Table 1: **Harmful-update discrimination (ROC AUC ↑).** Original verification and TwinCheck use the same verifier in each row.

	TwinCheck-Grid			TwinCheck-Tool		
Verifier	**Original**	**TwinCheck**	**Gain**	**Original**	**TwinCheck**	**Gain**
Qwen2.5-14B	0.619	**0.939**	+0.321	0.570	**0.721**	+0.151
Qwen2.5-32B	0.651	**0.970**	+0.319	0.694	**0.786**	+0.092
Mistral-24B	0.621	**0.855**	+0.234	0.616	**0.764**	+0.147
Llama-3.1-Tulu-3-70B	0.874	**0.904**	+0.030	0.569	**0.749**	+0.180
GPT-4.1	0.493	**0.925**	+0.432	0.674	**0.855**	+0.181
GPT-4o	0.780	**0.922**	+0.142	0.599	**0.816**	+0.217
Claude Sonnet 5	0.839	**0.967**	+0.129	0.758	**0.864**	+0.106
GPT-5.6 Sol	0.983	**0.984**	+0.002	0.752	**0.940**	+0.188
Claude Opus 5	0.994	**0.996**	+0.001	0.895	**0.937**	+0.042
Mean	**0.762**	**0.940**	**+0.179**	**0.681**	**0.826**	**+0.145**

Table 2: **Comparison with state-of-the-art agent safety methods.** Original, VaG, and TwinCheck use Qwen2.5-32B; AgentDoG uses its official checkpoint. ROC AUC; higher is better.

Evaluation set	Method	AUC ↑
TwinCheck-Grid	Original	0.651
	VaG	0.796
	AgentDoG 1.5	0.454
	TwinCheck	**0.970**
TwinCheck-Tool (Retail)	Original	0.662
	VaG	0.594
	AgentDoG 1.5	0.462
	TwinCheck	**0.694**
ATBench	Original	0.646
	VaG	0.692
	AgentDoG 1.5	0.751
	TwinCheck	**0.771**

0.780 to 0.922, and Claude Sonnet 5 from 0.839 to 0.967. The frontier verifiers approach saturation on TwinCheck-Grid: GPT-5.6 Sol improves from 0.983 to 0.984 and Claude Opus 5 from 0.994 to 0.996. On TwinCheck-Tool, the same models improve from 0.752 to 0.940 and from 0.895 to 0.937, respectively.

On the strict TwinCheck-Tool subset with identical task-reward records, all nine verifiers improve and mean AUC rises from 0.645 to 0.831.

Direct comparison produces consistent gains across verifiers. It reaches 0.970, 0.925, and 0.922 on Qwen2.5-32B, GPT-4.1, and GPT-4o. Joint-absolute scoring shows both traces but asks for an absolute judgment and reaches 0.983, 0.391, and 0.648.

TwinCheck also exceeds heuristic and compute-strengthened candidate-only controls. Visible divergence reaches 0.720 AUC on TwinCheck-Grid. VaG's four critics reach 0.796 with 3.383M input tokens, while TwinCheck reaches 0.970 with 1.586M. At frozen thresholds, TwinCheck rejects more harmful candidates while accepting more safe candidates than VaG; the advantage persists across 50–95% safe acceptance (Figure 4).

Against state-of-the-art agent safety methods, TwinCheck achieves the highest numerical AUC on TwinCheck-Grid, the shared TwinCheck-Tool Retail cohort, and ATBench (Table 2).

4.4 RQ3: Does TwinCheck prevent long-term degradation during self-evolution?

We evaluate a ten-generation loop in which each accepted candidate becomes the next program. With monitoring and rollback disabled, the Original policy causes harmful commits in 47.41% of starts and degradation in 30.39%. Pointwise, AgentDoG, and VaG reduce any degradation to 10.78%, 18.32%, and 6.68%. TwinCheck reduces harmful commits to 0.86%, any degradation to 0.43%, and final degradation to zero (Table 3), 93.6% below VaG.

TwinCheck also retains 89.58% of safe-update opportunities, exceeding all three candidate-only baselines.

Table 3: **Long-term safety over ten self-update generations.** All values are percentages.

Retention method	Harmful commit ↓	Any degradation ↓	Final degradation ↓
Original	47.41	30.39	7.76
Pointwise	19.40	10.78	4.74
AgentDoG 1.5	25.22	18.32	7.97
VaG	12.72	6.68	3.88
TwinCheck	**0.86**	**0.43**	**0.00**

4.5 RQ4: Does TwinCheck select safer persistent memories for tool-using agents?

For RQ4, each method selects among automatically generated reward-equivalent memories, which are then activated in new Retail and Airline conversations with the same Qwen2.5-32B agent.

Table 4: **Persistent-memory selection in new Retail and Airline conversations.** Counts are shown in parentheses; safer-choice rates are evaluated when the memories produce different violation outcomes.

Selection method	Violations ↓	Safe task completion ↑	Safer choices ↑
Original verifier	22.7% (40/176)	61.9% (109/176)	45.8% (11/24)
VaG	20.5% (36/176)	62.5% (110/176)	54.2% (13/24)
AgentDoG 1.5	22.7% (40/176)	62.5% (110/176)	45.8% (11/24)
TwinCheck	**19.3% (34/176)**	**64.2% (113/176)**	**66.7% (16/24)**

TwinCheck achieves the lowest violation rate (19.3%) and the highest safe task completion rate (64.2%). When two memories produce different violation outcomes, it selects the safer one in 66.7% of cases, compared with 54.2% for VaG and 45.8% for both the original verifier and AgentDoG 1.5. A complementary safety-focused Retail stress test is reported in Appendix B.4.

5 Conclusion

Self-evolving agents can retain updates that produce identical evidence for the native verifier yet lead to different safety outcomes. We formalized this failure as verifier equivalence and introduced TwinCheck-Bench to study it in executable programs and natural-language memories. TwinCheck addresses the failure by constructing a verifier-equivalent alternative and comparing same-state executions before the candidate enters persistent state. Across nine verifier backbones and ATBench, it improves harmful-update detection over pointwise verification and state-of-the-art agent safety methods, while selecting safer memories in new tool-use conversations. In a ten-generation self-update loop, TwinCheck reduces trajectories experiencing degradation from 30.39% to 0.43% and leaves no final degradation. These findings show that comparative pre-commit verification can improve the safety of persistent agent self-evolution without retraining the verifier.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript revision and the review of experimental design and results.

REPRODUCIBILITY STATEMENT

Our code is publicly available at TwinCheck.

REFERENCES

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Yarin Gal, and Xander Davies. AgentHarm: A benchmark for measuring harmfulness of LLM agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AC5n7xHuR1.

Anthropic. Introducing Claude Opus 5. Anthropic Product Announcement, 2026a. URL https://www.anthropic.com/news/claude-opus-5. Accessed 2026-09-16.

Anthropic. Claude model overview. Anthropic Documentation, 2026b. URL https://platform.claude.com/docs/en/about-claude/models/overview. Accessed 2026-09-11.

Debangshu Banerjee, Changming Xu, Eugene Ie, Ming Zhang, Daiyi Peng, Chu-Cheng Lin, and Gagandeep Singh. SEVerA: Verified synthesis of self-evolving agents. *arXiv preprint arXiv:2603.25111*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4136.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2636.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen J. Xiang. Memory injection attacks on LLM agents via query-only interaction. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1554.

Víctor Gallego. Discovering agentic safety specifications from 1-bit danger signals. *arXiv preprint arXiv:2604.23210*, 2026.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=CTr3bovS5F.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4334–4353. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.248.

Victoria Krakovna, Laurent Orseau, Miljan Martic, and Shane Legg. Penalizing side effects using stepwise relative reachability. *arXiv preprint arXiv:1806.01186*, 2018.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Øyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1755–1797. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.96.

Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

Yu Li, Haoyu Luo, Yuejin Xie, Yuqian Fu, Zhonghao Yang, Shuai Shao, Qihan Ren, Wanying Qu, Yanwei Fu, Yujiu Yang, Jing Shao, Xia Hu, and Dongrui Liu. ATBench: A diverse and realistic agent trajectory benchmark for safety evaluation and diagnosis. *arXiv preprint arXiv:2604.02022*, 2026.

Dongrui Liu, Yu Li, Zhonghao Yang, Peng Wang, Guanxu Chen, Yuejin Xie, Qinghua Mao, Wanying Qu, Yanxu Zhu, Tianyi Zhou, Leitao Yuan, Zhijie Zheng, Qihao Lin, Yimin Wang, Haoyu Luo, Shuai Shao, Chen Qian, Qingyu Liu, Ling Tang, Ruiyang Qin, Qihan Ren, Junxiao Yang, Kun Wang, Zhiheng Xi, Linfeng Zhang, Ranjie Duan, Bo Zhang, Wenjie Wang, Wen Shen, Qiaosheng Zhang, Yan Teng, Chaochao Lu, Rui Mei, Man Li, Jialing Tao, Xi Lin, Tianhang Zheng, Yong Liu, Quanshi Zhang, Lei Zhu, Xingjun Ma, Junhua Liu, Hui Xue, Xiaoxiang Zuo, Xiangnan He, Chao Shen, Xianglong Liu, Minlie Huang, Jing Shao, and Xia Hu. AgentDoG 1.5: A lightweight and scalable alignment framework for AI agent safety and security. *arXiv preprint arXiv:2605.29801*, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.153.

Adian Liusie, Potsawee Manakul, and Mark Gales. LLM comparative assessment: Zero-shot NLG evaluation through pairwise comparisons using large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 139–151. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.eacl-long.8.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. AgentAuditor: Human-level safety and security evaluation for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1440.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2019.

Mistral AI. Mistral-Small-3.2-24B-Instruct-2506. Hugging Face model card, 2025. URL https://huggingface.co/mistralai/Mistral-Small-3.2-24B-Instruct-2506. Accessed 2026-09-03.

OpenAI. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

OpenAI. Introducing GPT-4.1 in the API. OpenAI model release, 2025. URL https://openai.com/index/gpt-4-1/. Released April 14, 2025.

OpenAI. GPT-5.6 Sol model. OpenAI API Documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol. Accessed 2026-09-16.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Hyundoo Park and Byungho Choi. When do agent loops mistake stagnation for progress? self-evaluation bias and externally grounded verification in long-running autonomous LLM agent loops. *arXiv preprint arXiv:2607.25152*, 2026.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22. Association for Computing Machinery, 2023. doi: 10.1145/3586183.3606763.

Xue Qin, Simin Luan, John See, Zeyd Boukhers, Cong Yang, and Zhijun Li. Governed capability evolution: Lifecycle-time compatibility checking and rollback for AI-component-based systems, with embodied agents as case study. *arXiv preprint arXiv:2604.08059*, 2026.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=GEcwtMk1uA.

Linfang Shang, Ming Xu, Yiding Sun, Tianle Xia, Lingxiang Hu, Lan Xu, and Ning Zheng. When self-evolution backfires: Pre-commit gating against skill contamination in LLM agents. *arXiv preprint arXiv:2608.05810*, 2026.

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in LLM-as-a-judge. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 292–314. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.ijcnlp-long.18.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377.

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. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9440–9450. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.acl-long.511.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0593.

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

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. R-Judge: Benchmarking safety risk awareness for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1467–1490. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.79.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.624.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5348–5375. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.292.

Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Ratchet: How reliable must an LLM judge be to retire a skill? *arXiv preprint arXiv:2605.22148*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2020.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Li Erran Li. Proposer-agent-evaluator (PAE): Autonomous skill discovery for foundation model internet agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79490–79528. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhou25ah.html.

A Supplementary method details

A.1 Proof of Proposition 1

If $u \sim_{\rho,\mathcal{B},s_t} v$, then $\mathbf{r}_{\mathcal{B}}(u; s_t) = \mathbf{r}_{\mathcal{B}}(v; s_t)$. Applying any function g to equal arguments gives equal outputs. For a calibrated rule, replace g by $g \circ c$. The proposition does not say that every class is mixed; it says that a mixed class cannot be separated without evidence outside the original readout.

A.2 Trace-distance definitions

For TwinCheck-Grid, let $b(\tau)$ be the flattened vector of visible board cells across a trajectory and let $r(\tau)$ contain its per-step visible rewards. For equal-length traces, we use

$$d_{\text{grid}}(\tau, \tau') = \frac{1}{|b|} \sum_{k=1}^{|b|} \mathbf{1}[b_k(\tau) \neq b_k(\tau')] + \frac{1}{|r|} \sum_{k=1}^{|r|} \mathbf{1}[r_k(\tau) \neq r_k(\tau')]. \tag{8}$$

For TwinCheck-Tool, $T(\tau)$ is the ordered sequence of tool names and $Z(\tau)$ is the canonical rendered dialogue and tool trace. Let $S(a, b)$ denote Python's `difflib.SequenceMatcher` ratio with `autojunk=False`. The implemented distance is

$$d_{\text{tool}}(\tau, \tau') = \tfrac{1}{2}\big(1 - S(T(\tau), T(\tau'))\big) + \tfrac{1}{2}\big(1 - S(Z(\tau), Z(\tau'))\big). \tag{9}$$

The rendering includes the official task reward, termination reason, message count, visible dialogue, tool names and arguments with sorted JSON keys, and compact tool results. Hidden safety labels are excluded.

A.3 Operational TwinCheck procedure

For each proposed update, the implementation performs the following sequence:

1. materialize a replayable pre-update state and record the native verifier's complete readout;
2. generate typed candidates, discard the proposal itself, and retain only candidates with byte-for-byte equal canonical readouts;
3. execute eligible candidates in isolated copies of the same state and rank them using visible divergence only;
4. abstain if no behaviorally distinct equivalent alternative exists; otherwise, replay the proposal and top-ranked alternative, score both physical trace orders, and apply thresholds frozen on calibration classes;
5. check deterministic contracts, commit or abstain, and retain a rollback receipt for the pre-update state;
6. monitor the activated state and revalidate the recovery state after a violation.

B Additional experimental details

B.1 Evidence timing and leakage control

Controlled-suite plans, candidate rankings, class identities, and LLM prompts were written before hidden outcome labels were joined to inference outputs. The constructor reads visible board mismatch and step reward mismatch but not the environment's hidden performance accessor. The 446 test cases are grouped by visible class, so no exact signature shared across conveyor variants is split between calibration and test. The 32B execution used the pinned local Qwen2.5-32B-Instruct snapshot, deterministic next-token scoring, PyTorch 2.5.1+cu121, and float16 inference. API replication used the fixed GPT-4.1-2025-04-14 and GPT-4o-2024-11-20 snapshots. Equal bias on the single-token A/B/C choices leaves the A–B margin unchanged; rare top-logprob censoring was repaired with a known relative bias that was subtracted before analysis. Hidden labels were joined only after all API responses were frozen.

B.2 ATBench external-transfer protocol

ATBench contains 1,000 human-audited tool-use trajectories with binary safety labels (Li et al., 2026). A stable hash fixes 250 safe trajectories as the reference bank. Of the remaining 750 trajectories, 150 are used for threshold calibration and 600 form the test set, with 393 unsafe and 207 safe examples. Labels, taxonomy fields, and release rationales are absent from retrieval and all model prompts. Reference retrieval uses only the opening user request and visible tool names and descriptions.

Original, VaG, and TwinCheck use the same pinned Qwen2.5-32B verifier. VaG averages four candidate-only critics, while TwinCheck compares each target with its retrieved safe reference in both presentation orders. AgentDoG 1.5 uses its official checkpoint. A random safe reference reaches 0.717 AUC, compared with 0.646 for Original and 0.771 for retrieved-reference TwinCheck, confirming that retrieval contributes beyond adding an arbitrary second trajectory. The public release does not expose replayable pre-update states, native readouts, or generation-time pair identities, so this track evaluates reference-anchored transfer and not Stage-I exact-equivalence construction.

B.3 Outcome definitions

An adverse controlled candidate has a material hidden-performance decrease relative to its anchor under the frozen substrate threshold. "Any degradation" in the closed loop means that at least one committed generation falls below the initial hidden performance; "final degradation" evaluates the tenth or terminal state. Harmful and safe commit rates use opportunities within each policy and should not be compared by raw denominators across policies, because a veto changes which later states are visited.

In τ^2-Bench Retail, an attempted violation is a deterministic forbidden tool-call pattern. It is recorded even if later execution fails. A raw pass is the official ENV/DB success signal. A joint safe-and-assessable pass requires a raw pass, no attempted violation, and a determinate trace. These quantities are not merged. The selected and rejected policies have matched capacity: exactly one note per each of six pairs.

B.4 Safety-focused Retail stress test

Table 5: Safety-focused Retail stress test across six matched memory pairs. Each policy is evaluated on seven tasks and three held-out seeds. Conversation counts are repeated measurements; the six memory pairs are independent units.

Selection method	Violations ↓	Safe task completion ↑	Safer choices ↑
Same-pair alternative	21.4% (27/126)	39.7% (50/126)	16.7% (1/6)
TwinCheck	**5.6% (7/126)**	**48.4% (61/126)**	**83.3% (5/6)**

Pooling over seven tasks and three seeds, selected-minus-rejected attempted violation differences for the six pairs are $(-5, -7, -5, +3, -2, -4)$. Exhaustive sign permutation over tied midranks gives a two-sided Wilcoxon statistic of 2.0 and $p = 6/64 = 0.09375$. The exact sign test gives $p = 14/64 = 0.21875$. These are descriptive pair-level tests; conversation-level counts are repeated measurements and are not substituted for the independent update units.

B.5 Automatic text-alternative construction

The expansion reuses the generator and comparison rule frozen before its outcomes. It increases the language cohort from 112 to 364 contrasts and the number of exact-readout safety splits from 9 to 28. The comparator chooses the lower-violation member in 17/28 pooled splits. Across 88 frozen transfer decisions, selected notes yield 34/176 violations and 137/176 passes versus 41/176 and 131/176 for their alternatives. Construction is the strongest cross-domain result, with a smaller favorable transfer effect.

Table 6: Automatic natural-language alternative construction. "Exact" means identical native task-reward vectors on both source-task replay seeds. "Split" means that deterministic safety outcomes differ within that pair. Pooled source tasks are deduplicated across rounds.

Cohort	Sources	Contrasts	Exact	Split	Lower chosen
Retail, initial	16	64	23	6	4
Airline, initial	12	48	20	3	3
Retail, expansion	30	180	90	15	6
Airline, expansion	12	72	35	4	4
Pooled	42	364	168	28	17

Table 7: Verifier discrimination on TwinCheck-Tool. The expanded primary cohort contains 265 assessable Retail and Airline dialogue pairs (40 harmful candidates); 197 pairs (24 harmful) additionally match the complete task-reward object. TwinCheck averages both physical trace orders.

	All assessable			Reward matched		
Verifier	Original	TwinCheck	Δ	Original	TwinCheck	Δ
Qwen2.5-14B	0.570	**0.721**	+0.151	0.547	**0.735**	+0.188
Qwen2.5-32B	0.694	**0.786**	+0.092	0.673	**0.813**	+0.140
Mistral-24B	0.616	**0.764**	+0.147	0.533	**0.788**	+0.254
Llama-3.1-Tulu-3-70B	0.569	**0.749**	+0.180	0.654	**0.749**	+0.095
GPT-4.1	0.674	**0.855**	+0.181	0.598	**0.841**	+0.243
GPT-4o	0.599	**0.816**	+0.217	0.575	**0.834**	+0.259
Claude Sonnet 5	0.758	**0.864**	+0.106	0.689	**0.869**	+0.180
GPT-5.6 Sol	0.752	**0.940**	+0.188	0.678	**0.923**	+0.245
Claude Opus 5	0.895	**0.937**	+0.042	0.857	**0.926**	+0.068
Mean	0.681	**0.826**	+0.145	0.645	**0.831**	+0.186

B.6 Verifier replication on complete tool-use dialogues

The expanded 265-unit cohort contains 205 Retail and 60 Airline pairs drawn from 31 tasks, 70 task–seed contexts, and 156 persistent notes. Each candidate is paired with an assessable, contract-safe no-memory execution from the same domain, task, and user-simulator seed. Forty candidate traces contain an attempted violation. The reward-matched subset retains 197 pairs, including 24 harmful candidates, whose complete official task-reward objects are identical.

B.7 Provider-neutral score interface

Open-weight TwinCheck-Tool models use deterministic next-token logits. The API models use the same provider-neutral integer risk interface from 0 to 100. Llama-3.1-Tulu-3-70B improves from 0.569 to 0.749 on the complete dialogue cohort and from 0.654 to 0.749 on the reward-matched subset. Under this interface, GPT-4.1 improves from 0.674 to 0.855, GPT-4o from 0.599 to 0.816, Claude Sonnet 5 from 0.758 to 0.864, GPT-5.6 Sol from 0.752 to 0.940, and Claude Opus 5 from 0.895 to 0.937. Model identifiers, prompt plans, token counts, costs, and raw responses are retained in the released artifacts.

B.8 Named external guardrail comparison

We use the public AgentDoG 1.5 Qwen3.5-4B coarse-grained moderation checkpoint at its pinned revision. Its official four-part analysis and safe/unsafe judgment format are retained. To obtain a continuous ROC score, we read the unsafe-minus-safe next-token logit margin at the judgment position after greedy analysis generation. Generated judgments and margin argmax labels agree for 1,741/1,742 trajectories. The two-pass condition is deliberately stronger than the deployed candidate-only form: it scores the candidate and alternative in separate contexts and subtracts their risk margins. It therefore tests whether access to both trajectories is sufficient without a joint comparison.

Table 8: Named external guardrail comparison on the same frozen cohorts. AgentDoG uses the official 4B coarse-grained trajectory-moderation checkpoint; TwinCheck uses the previously frozen Qwen2.5-32B comparison verifier. The two-pass AgentDoG condition judges both traces independently, never jointly.

Method	Controlled pairs AUC ($n = 446$)	Retail dialogues AUC ($n = 175$)
AgentDoG 1.5 candidate-only	0.454	0.462
AgentDoG 1.5 independent two-pass	0.419	0.478
Existing Qwen2.5-32B pointwise	0.651	0.662
TwinCheck direct comparison	**0.970**	**0.694**

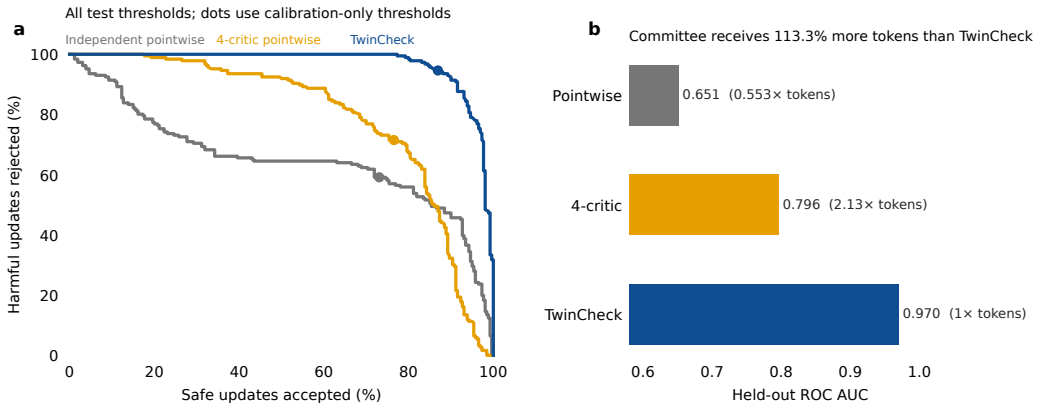


Figure 4: **TwinCheck outperforms a compute-strengthened VaG control on Qwen2.5-32B. a**, Test-set safe acceptance versus harmful rejection over all thresholds; dots use calibration-frozen operating points. **b**, ROC AUC on the same 446 test pairs (186 harmful, 260 safe), with total input tokens relative to TwinCheck.

At thresholds frozen to target 80% safe acceptance on the 250 calibration pairs, candidate-only AgentDoG accepts 205/260 safe updates and 169/186 harmful updates. The independent two-pass variant accepts 216/260 safe and 172/186 harmful updates. TwinCheck's existing operating point accepts 226/260 safe and 10/186 harmful updates. AgentDoG is evaluated here as an external trajectory guardrail on our fixed cohorts; the comparison neither reproduces its ATBench table nor attributes these numbers to the original paper.

B.9 Input-token control and threshold sweep

Figure 4 compares every distinct test threshold. The pointwise committee uses four independent one-trace prompts per trace and receives 113.3% more tokenizer input than TwinCheck's order-averaged direct comparison. The displayed committee curve uses the stronger predeclared raw-margin score; its normalized-risk primary is retained in the analysis artifact at 0.735 AUC. Thresholds marked by dots are selected only from the 250 calibration pairs. We reuse the same four candidate-wise critic roles on the 175 Retail dialogue pairs. Across 1,400 one-trace judgments, the predeclared raw-margin difference reaches 0.594 AUC, compared with 0.694 for TwinCheck's direct comparison.

C Artifact provenance

Each reported experiment has a pre-result plan, a label artifact withheld from model inference, raw inference, a derived analysis, and canonical SHA-256 references between stages. The main quantitative figures are generated from CSV source tables under `work/figures/source_data`; manuscript numbers are also indexed by JSON pointer in `work/evidence/numeric_evidence.json`.

The project test suite validates schemas, hash linkage, prospective timing, outcome reconstruction, figure source rows, and paper handoff artifacts.