LONGTOOL-BENCH: BENCHMARKING LIFECYCLE-AWARE ADAPTATION IN LONG-HORIZON TOOL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon tool agents execute interdependent actions that modify persistent application state, often across applications. Yet tool capabilities, interfaces, permission boundaries, execution conditions, or identities may change after consequential effects have been committed. Successful adaptation must revise invalid assumptions while preserving valid progress, avoiding duplicate effects, and satisfying revised constraints. Existing benchmarks study long-horizon execution, stateful interaction, or unreliable tools, but rarely combine controlled in-episode evolution, retained side effects, and executable recovery obligations within an unfinished task. We introduce LONGTOOL-BENCH, comprising 200 AppWorld tasks, 400 evolution contracts, and 38,400 episodes across sixteen models. It injects changes at semantically eligible boundaries, preserves committed state, and varies access to change information while separately measuring task correctness, event exposure, contract recovery, prohibited behavior, recovery effort, and declaration reliability. Model-independent executable witnesses establish chain feasibility from retained boundary states. Relative to Hidden, Oracle-Docs improves event-chain resolution by 23.9 pp but Official Success by only 5.0 pp; moreover, 86.8% of chain-resolved episodes still fail the original task. Category analyses and worker ablations further separate failures in exposure, recovery, state protection, and terminal completion. These results show that contract recovery alone does not ensure end-to-end adaptation: dependable agents must jointly track evolving contracts, committed state, unfinished dependencies, and terminal task evidence.

1 INTRODUCTION

Large language model agents rely on tools to ground decisions in external environments, inspect state, and execute consequential actions (Yao et al., 2022; Schick et al., 2023; Shinn et al., 2023). Agent evaluation consequently spans settings from isolated API calls to interactive web and computer environments and multi-application workflows (Li et al., 2023; Qin et al., 2024; Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024). Success requires coordinating interdependent actions while tracking how earlier tool operations alter persistent state and constrain later decisions.

Tool evolution is consequential in long-horizon workflows: early assumptions may fail after interdependent actions modify state. Interfaces may change, permissions expire, or endpoints be retired. Conflicts between revised tool contracts and accumulated state constrain recovery: replaying completed operations can duplicate effects, while migrating to a replacement API can invalidate downstream dependencies. This raises the central question: ***Can an agent adapt to an in-episode tool change while preserving committed progress and completing the long-horizon task?***

Existing benchmarks study stateful interaction (Lu et al., 2025; Yao et al., 2024; Barres et al., 2025), stabilized access to unreliable APIs (Guo et al., 2024), runtime failures and replanning (Tian et al., 2026; Xi et al., 2026; Zhu et al., 2026; Liu et al., 2026b;c), or cross-version tool evolution (Liu et al., 2026a). These lines leave underexplored the joint setting of contract changes within unfinished workflows, preservation of committed effects, and executable recovery obligations. This gap obscures whether failures reflect contract recovery or state preservation.

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.

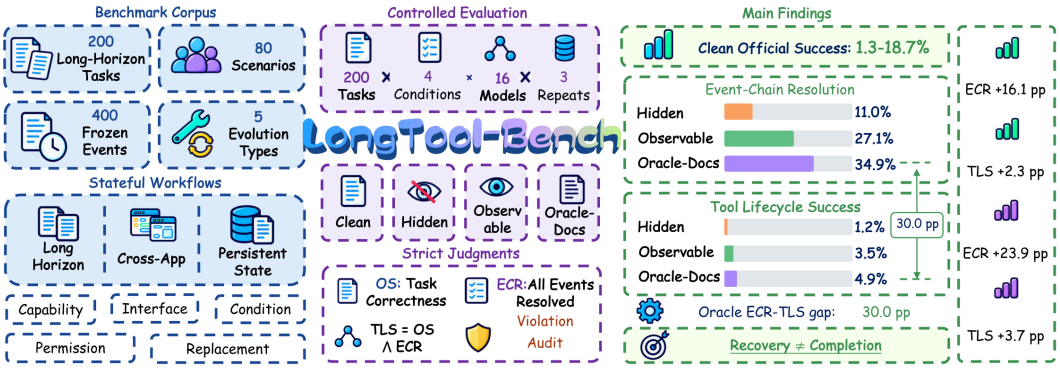

Figure 1: **Overview of LONGTOOL-BENCH.** LONGTOOL-BENCH combines long-horizon tasks, five forms of tool evolution, and persistent state with task-, lifecycle-, and trajectory-level evaluation.

To address this problem, we introduce LONGTOOL-BENCH for *lifecycle-aware adaptation* in long-horizon tool agents. Built on stateful AppWorld workflows, it instantiates controlled capability, interface, permission, execution-condition, and replacement changes while retaining the task objective (Figure 1). Events activate at eligible semantic boundaries in each trajectory, interrupt triggering calls before execution, and preserve committed state. Executable contracts define verifiable requirements for plan revision, authorization, compensation, or tool migration before execution resumes.

Matched conditions vary tool evolution and change-information access while fixing the task, initial state, worker, evaluator, and budget; model behavior determines exposure and activation timing. Across sixteen models, richer change information improves full-chain resolution, yet joint lifecycle and task success remains low. This gap shows contract resolution does not ensure task completion. Ledgers record prohibited behavior and recovery effort, support comparisons between completion declarations and terminal evaluator verdicts, and enable trace-level diagnosis without estimating failure prevalence.

Our contributions are threefold:

- **A long-horizon tool-evolution benchmark.** We construct 400 controlled events across 200 long-horizon tool-use tasks with persistent application state, committed side effects, and unresolved downstream dependencies.
- **A contract-grounded intervention protocol.** Trajectory-dependent triggers interrupt calls before their effects; executable witnesses establish feasibility from retained boundary states, and matched conditions share the task, initial state, worker, and evaluator.
- **A ledger-backed evaluation.** We report task correctness, exposure, conditional recovery, chain success, recorded violations, recovery effort, and declaration reliability to distinguish failures across activation, adaptation, and terminal verification.

2 RELATED WORK

Tool-Using Agents. ReAct and Reflexion combine action selection with feedback; Toolformer and ToolACE study invocation and call synthesis (Yao et al., 2022; Shinn et al., 2023; Schick et al., 2023; Liu et al., 2025). API benchmarks evaluate tool selection and multi-step execution (Li et al., 2023; Qin et al., 2024; Patil et al., 2025; Ye et al., 2025; Lei et al., 2025). AgentBench, WebArena, and OSWorld cover interactive environments (Liu et al., 2024; Zhou et al., 2024; Xie et al., 2024). Most assume fixed interfaces; LONGTOOL-BENCH changes contracts after state changes accumulate.

Long-Horizon Agent Benchmarks. HORIZON uses controlled horizons to measure degradation over longer action sequences (Wang et al., 2026). Odysseys and OSWorld 2.0 evaluate long-horizon web and computer-use workflows across applications and changing state (Jang et al., 2026; Yuan et al., 2026). Long-Horizon-Terminal-Bench evaluates terminal execution with dense intermediate rewards over engineering and scientific tasks (Li et al., 2026). These benchmarks characterize planning, memory, verification, and efficiency over long trajectories but hold tool contracts fixed.

Table 1: **Representative tool-agent benchmarks spanning tool use, stateful interaction, and dynamic tool environments.** PS = persistent mutable state; CC = controlled change; IE = change within an unfinished task; RA = separately scored recovery outcome; BA = unintended-effect or prohibited-operation audit. Marks indicate explicit properties or metrics; scales use native units.

Benchmark	State	Evolution		Evaluation		Reported Scale
	PS	CC	IE	RA	BA	
ToolLLM (Qin et al., 2024)	✗	✗	✗	✗	✗	126,486 instruction–solution pairs; 16,464 APIs
ToolSandbox (Lu et al., 2025)	✓	✗	✗	✗	✓	1,032 test cases; 34 tools
AppWorld (Trivedi et al., 2024)	✓	✗	✗	✗	✓	750 tasks; 457 APIs / 9 apps
τ-bench (Yao et al., 2024)	✓	✗	✗	✗	✗	165 tasks; 28 API tools
ScrambleToolBench (Toh et al., 2026)	✗	✓	✗	✓	✗	28 core tools; 3 dynamic settings
ToolBench-X (Tian et al., 2026)	✗	✓	✓	✗	✗	1,106 tasks; 4,956 tools
C-World (Xi et al., 2026)	✓	✓	✓	✓	✗	5,571 tools / 204 apps
ToolMaze (Zhu et al., 2026)	✗	✓	✓	✓	✗	2,000 instances; 270 tools
EvoArena (Terminal) (Xu et al., 2026)	✓	✓	✗	✓	✗	89 seeds; 352 evolved versions
LONGTOOL-BENCH (ours)	✓	✓	✓	✓	✓	**200 tasks / 80 families; 400 events / 5 categories**

Tool Use under Dynamic Conditions. StableToolBench caches online responses for reproducible API evaluation (Guo et al., 2024). C-World injects controlled failures and reports recovery; ToolBench-X guarantees recoverability but reports task completion rather than a separate recovery outcome; ToolMaze evaluates recovery and cost (Xi et al., 2026; Tian et al., 2026; Zhu et al., 2026). ScrambleToolBench studies cross-task hidden mapping drift within continuous episodes, whereas EvoArena evaluates persistent version chains with oracle solvability checks (Toh et al., 2026; Xu et al., 2026). Outcome Monitors supplies nonbinding recovery feedback, while LedgerAgent enforces ledger-grounded policies (Panthi & Abdelfattah, 2026; Uddin et al., 2026). PlanBench-XL changes viable paths, CostBench changes execution costs, and MCPEvol-Bench compares server versions (Liu et al., 2026b;c;a). Together, these benchmarks distinguish unstable execution, recoverable perturbations, and cross-version drift, but do not combine them with retained within-task state or evaluate contract compliance separately from terminal success. In contrast, LONGTOOL-BENCH combines retained effects with within-task contract changes, establishes end-to-end task feasibility through executable witnesses, and scores terminal correctness independently from prohibited behavior.

3 LONGTOOL-BENCH

LONGTOOL-BENCH evaluates contract-conformant adaptation when tools evolve within unfinished long-horizon workflows after prior effects have been committed. Figure 2 summarizes stateful task construction, executable contracts, and matched online evaluation.

3.1 STATEFUL TASK CONSTRUCTION

We construct LONGTOOL-BENCH from AppWorld, where agents complete stateful tasks through executable APIs that read and modify persistent database and shell state (Trivedi et al., 2024). Each task is $\mathcal{I} = (q, x_0, \mathcal{T}_0, \mathcal{A})$, where q is the user instruction, x_0 the initial environment state, $\mathcal{T}_0$ the initial tool environment, and $\mathcal{A}$ the original state-based assertions. We retain $\mathcal{A}$ unchanged, evaluating adapted and clean trajectories against the same task specification.

Task curation combines long-horizon screening with reference-trace analysis. Selection requires extended execution, non-adjacent call dependencies, and persistent side effects that constrain later decisions; 120 workflows additionally require cross-application coordination. For each task, a reference execution induces a dependency graph $G_q = (V_q, E_q)$ whose nodes are tool calls and whose edges encode data or state dependencies. State-changing calls with downstream dependents define candidate evolution regions; at each such boundary, a pre-call fingerprint ϕ_t records state and tool context. Reference executions establish task feasibility and the screening horizon; realized actor turns and API attempts are recorded separately. Core-200 pairs these long-horizon AppWorld workflows with controlled tool changes triggered at validated semantic boundaries during execution.

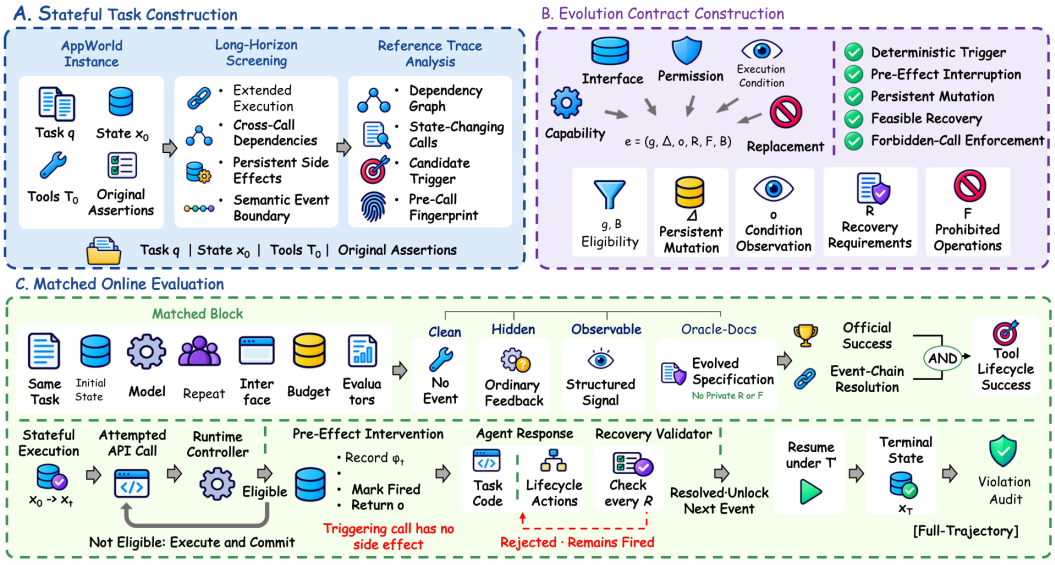

Figure 2: **LONGTOOL-BENCH protocol.** (A) Stateful task construction, (B) executable evolution contracts, and (C) matched online evaluation preserving committed effects and resulting state.

3.2 EVOLUTION CONTRACT CONSTRUCTION

The benchmark covers capability, interface, permission, execution condition, and replacement changes. Each event has a frozen contract $e_k = (g_k, \Delta_k, o_k, \mathcal{R}_k, \mathcal{F}_k, \mathcal{B}_k)$. The predicate $g_k(x_t, \mathcal{T}_t, h_t, \mathcal{L}_t, a_t)$ encodes the event-specific trigger before candidate call a_t, conditioned on current state x_t, active tools $\mathcal{T}_t$, execution history h_t, and lifecycle ledger $\mathcal{L}_t$. The operator Δ_k specifies the persistent mutation, $o_k(c)$ the condition-specific feedback, $\mathcal{R}_k$ the recovery predicates, $\mathcal{F}_k$ the prohibited operations, and $\mathcal{B}_k$ the admissible boundary signatures. Separating condition-specific observations from fixed mutation, trigger, recovery, and audit semantics anchors evolution to semantic workflow boundaries rather than fixed turns or reference-trace indices.

Each contract undergoes executable validation before inclusion. A valid event must activate deterministically under its eligibility rule, interrupt the triggering call before any side effect, persistently apply $\Delta_k(\mathcal{T}_t)$, and permit programmatic checks of $\mathcal{R}_k$ and $\mathcal{F}_k$. The finite set $\mathcal{B}_k$ restricts activation to $\phi_t \in \mathcal{B}_k$. For each validated boundary and reachable chain prefix, a model-independent witness starts from the retained state, resolves the remaining events, and satisfies the original assertions, establishing chain feasibility before model evaluation. Validated events form an ordered chain $\mathcal{E} = (e_1, \ldots, e_K)$ with states $z_k \in \{\text{PLANNED}, \text{FIRED}, \text{RESOLVED}\}$. Event e_k becomes FIRED when Δ_k is applied and RESOLVED only after all recovery predicates pass; e_{k+1} cannot activate until $z_k = \text{RESOLVED}$ (Appendix B).

3.3 MATCHED ONLINE EVALUATION

Matched conditions. Each task is evaluated under CLEAN, HIDDEN, OBSERVABLE, and ORACLE-DOCS within a matched block. Task, initial state, model alias, run-level repeat key, action interface, budget, stopping rule, and evaluators are fixed. CLEAN retains $\mathcal{T}_0$ and activates no event, whereas evolving conditions share the frozen chain $\mathcal{E}$. HIDDEN exposes ordinary evolved-environment feedback; OBSERVABLE adds a structured change signal; and ORACLE-DOCS supplies the authoritative evolved specification without revealing private predicates in $\mathcal{R}_k$ or $\mathcal{F}_k$. All evolving conditions use the same lifecycle interface; HIDDEN receives no event identity, category, required-action list, or validator payload. A schema-neutral control removes semantic lifecycle-action names while preserving the task interface.

At each attempted API call, the runtime controller first enforces prohibitions accumulated from activated contracts and then evaluates the next event's eligibility rule. If no next event is eligible, the

call executes normally and updates the environment. Otherwise, the controller records ϕ_t, applies $\mathcal{T}_{t^+} = \Delta_k(\mathcal{T}_t)$, sets $z_k = \text{FIRED}$, and returns observation $o_k(c)$. The triggering call is terminated before execution, yielding $x_{t^+} = x_t$ at the intervention boundary. All effects committed before t remain in the environment, so the agent must continue from the preserved state rather than restart the task.

Following intervention, the agent emits task code and structured lifecycle actions through separate channels. Task code proposes workflow continuation; lifecycle actions either invoke permitted recovery operations or submit machine-checkable evidence. The validator checks the submitted sequence, scope, tool identities, mappings, and state evidence against $\mathcal{R}_k$. Invalid or incomplete evidence leaves $z_k = \text{FIRED}$ and blocks workflow resumption while permitted recovery operations remain available. Once all encoded predicates pass, the event becomes RESOLVED, execution resumes under the evolved environment, and the next event can satisfy its eligibility rule. Resolution denotes compliance with the executable benchmark contract (Appendix C).

Evaluation criteria. The original AppWorld evaluator determines Official Success as $\text{OS} = \mathbb{1}[\bigwedge_{\alpha \in \mathcal{A}} \alpha(x_T)]$. For evolving episodes, let F and R denote fired and resolved event counts; conditional Lifecycle Recovery is $\text{LR} = R/F$ for $F > 0$, and Event-Chain Resolution is $\text{ECR} = \mathbb{1}[\forall e_k \in \mathcal{E},\, z_k = \text{RESOLVED}]$. OS measures route-agnostic task completion, ECR measures full-chain resolution, and Tool Lifecycle Success is their conjunction, $\text{TLS} = \text{OS} \wedge \text{ECR}$. A successful route that avoids the chain has $\text{OS} = 1$ and $\text{ECR} = 0$. These outcomes separate terminal correctness from lifecycle execution; witnesses establish joint feasibility from retained boundary states. The append-only ledger records calls, effects, lifecycle actions, validator decisions, and transitions, supporting audits of prohibited operations, retired-tool reuse, disallowed repeated effects, invalid migrations, and unresolved-contract bypasses (Appendix E).

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Dataset. Core-200 contains 200 tasks selected from a 250-task AppWorld candidate pool using model-independent criteria (Trivedi et al., 2024), spanning 80 scenario families and including 120 cross-application workflows. Selection requires persistent state changes, cross-call dependencies, and a reference solution of at least 80 API calls. Reference traces establish clean feasibility, reconstruct dependency graphs, and locate semantic intervention boundaries; agents may follow any route satisfying the original assertions. Each task has one to three ordered contracts, yielding 400 events balanced across five categories. Every chain passes model-independent replay from retained boundary states against the original assertions.

Table 2: **Core-200 statistics.** Counts describe the frozen task corpus and event allocation; task attributes may overlap.

Statistic	Count / allocation
Corpus scale	
Candidate task pool	250 tasks
Frozen evaluation set	**200 tasks**
Scenario families	**80**
Evolution contracts	**400 events**
Task composition	
Retention from candidate pool	200 / 250 (80.0%)
Cross-application workflows	120 (60.0%)
Minimum reference horizon	80 API calls
Horizon: 80–149 / 150–299 / $\geq$300	70 / 70 / 60
Evolution design	
Tasks with 1 / 2 / 3 events	40 / 120 / 40
Evolution categories	5
Contracts per category	80

Models. We evaluate eight open-weight models: Llama-3.3-70B, Gemma-3-27B, Qwen3-32B, Qwen3-235B-A22B, Qwen3-Next-80B-A3B, DeepSeek-V3.1, GLM-4.5, and Mistral-Small-3.1 (Meta, 2024; Google, 2025; Qwen Team, 2025c;a;b; DeepSeek-AI, 2025; Z.ai, 2025; Mistral AI, 2025b). The eight API-access models are GPT-5-mini, GPT-5.1, GPT-5.5, Claude Haiku 4.5, Claude Sonnet 4.5, Claude Opus 4.1, Gemini 2.5 Flash, and Gemini 2.5 Pro (OpenAI, 2025b;a; Anthropic, 2025c;b;a; Comanici et al., 2025). All main-matrix episodes use the shared worker. Separate standard AppWorld-worker Clean and schema-neutral evolving configurations are implementation diagnostics outside the main estimands. Appendix D specifies worker limits, condition controls, and repeat protocol.

Metrics and protocol. Official Success is $\text{OS} = \mathbb{1}[\bigwedge_{\alpha \in \mathcal{A}} \alpha(x_T)]$; Assertion Coverage (AC) is the fraction of original assertions satisfied at termination. For planned, fired, and resolved counts

Table 3: **Task completion and protocol outcomes on Core-200.** OS denotes Official Success; LR denotes conditional Lifecycle Recovery. Strict ECR jointly requires full-chain exposure and resolution; TLS additionally requires OS. Pooled ECR and TLS are equal-weight means over Hidden, Observable, and Oracle-Docs. Levels are in % (↑); $\Delta_{\mathrm{D-H}}$ is the unrounded Oracle-Docs–Hidden difference in pp. Bold marks the highest result within each group; underlining marks the highest point estimate overall.

Model	Task completion: OS					Conditional recovery: LR				Pooled outcomes	
	Clean	Hidden	Observ-able	Oracle-Docs	$\Delta_{\mathrm{D-H}}$	Hidden	Observ-able	Oracle-Docs	$\Delta_{\mathrm{D-H}}$	ECR	TLS
Open-weight models (8)											
Llama-3.3-70B	5.33	2.00	3.33	4.67	+2.67	35.42	52.00	61.54	+26.12	18.00	1.33
Gemma-3-27B	1.33	0.67	1.33	2.00	+1.33	26.00	50.00	60.00	**+34.00**	14.00	0.00
Qwen3-32B	4.67	1.33	2.00	2.67	+1.33	36.00	54.72	63.64	+27.64	20.00	0.00
Qwen3-235B-A22B	11.33	4.67	7.33	9.33	+4.67	39.62	59.65	70.00	+30.38	25.33	4.00
DeepSeek-V3.1	10.67	4.00	7.33	10.00	+6.00	42.31	62.50	71.67	+29.36	27.33	4.00
Mistral-Small-3.1	7.33	2.67	4.00	5.33	+2.67	35.42	52.08	60.42	+25.00	16.67	1.33
Qwen3-Next-80B-A3B	**14.67**	**6.00**	**10.67**	**13.33**	**+7.33**	**44.83**	**69.35**	**78.79**	+33.96	**32.67**	**6.00**
GLM-4.5	8.00	2.67	5.33	7.33	+4.67	36.00	56.60	66.67	+30.67	22.00	2.00
Proprietary/API-access models (8)											
Gemini 2.5 Flash	6.67	2.00	4.00	5.33	+3.33	32.65	54.90	63.64	+30.98	18.00	1.33
Gemini 2.5 Pro	9.33	2.67	5.33	7.33	+4.67	31.91	56.00	64.81	+32.90	17.33	1.33
Claude Haiku 4.5	7.33	2.00	4.67	6.67	+4.67	34.62	56.36	67.80	+33.18	22.67	2.00
Claude Sonnet 4.5	17.33	7.33	12.00	15.33	+8.00	42.37	68.75	79.41	+37.04	34.67	6.67
Claude Opus 4.1	13.33	4.67	8.67	11.33	+6.67	37.50	62.30	72.31	+34.81	28.00	4.00
GPT-5-mini	11.33	4.00	7.33	10.00	+6.00	39.29	61.02	71.43	+32.14	26.00	4.00
GPT-5.1	16.67	6.67	10.67	14.67	+8.00	40.35	65.57	75.38	+35.03	30.67	6.00
GPT-5.5	**18.67**	**8.00**	**12.67**	**16.67**	**+8.67**	**43.33**	**70.77**	**82.61**	**+39.28**	**36.00**	**7.33**

P, F, R, Fire Rate is $\mathrm{FR} = F/P$ and conditional Lifecycle Recovery $\mathrm{LR} = R/F$. Event-Chain Resolution (ECR) requires every planned event to fire and resolve; Tool Lifecycle Success is $\mathrm{TLS} = \mathrm{OS} \wedge \mathrm{ECR}$. OS, FR, LR, ECR, and TLS are condition-specific; Table 3 reports ECR and TLS as equal-weight means across evolving conditions. The protocol-violation rate is $\mathrm{UR} = U/X$ over opportunity episodes. RAC counts actions and API attempts among resolved events; DR measures declaration frequency, and CP measures terminal success among declaring episodes. The matrix contains $16 \times 200 \times 4 \times 3 = 38{,}400$ episodes. We use 10,000 family-clustered paired bootstrap replicates for reported confidence intervals. Appendix E defines estimands, aggregation, cluster-bootstrap uncertainty, conditional metric sets, and scope of all reported analyses.

4.2 Main Results

Table 3 reveals three linked bottlenecks. First, Hidden OS is below Clean for every model, while OS and LR increase monotonically with more change information. Oracle-Docs raises LR by 25.00–39.28 pp but OS by only 1.33–8.67 pp, leaving 15 of 16 models below their Clean baselines. This asymmetry is consistent with specifications addressing contract uncertainty more than downstream state-repair demands. Second, Qwen3-Next-80B-A3B leads open-weight ECR and TLS at 32.67% and 6.00%, respectively; GPT-5.5 leads overall at 36.00% and 7.33%. At the group level, higher Clean OS co-occurs with higher ECR and TLS, so lifecycle competence complements rather than replaces baseline planning. Gemma-3-27B illustrates this separation: its 34.00-pp LR gain accompanies zero TLS despite improved local recovery. Third, 7,008 of 28,800 evolving episodes attain ECR, yet only 924 also satisfy the original assertions: 86.8% of chain-resolved episodes fail the task. These joint outcomes show that contract resolution is insufficient for terminal success; the aggregate gap does not identify when the remaining failure arose.

4.3 Outcomes under Change Information

Figure 3 reveals a two-stage response to change information. Relative to Hidden, ECR increases by 16.1 and 23.9 pp under Observable and Oracle-Docs, respectively, whereas OS increases by

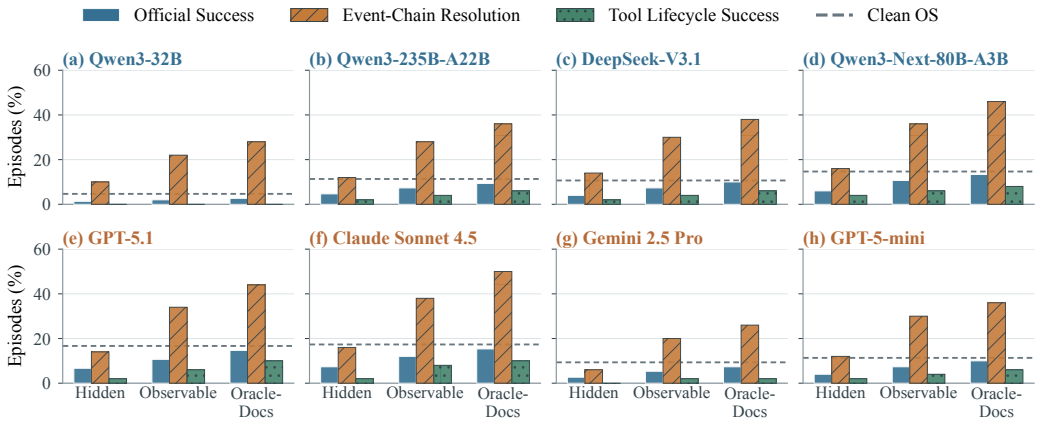

Figure 3: **Outcomes by change-information condition.** Bars show eight models spanning families and scales, balanced by access group; dashed lines mark model-specific Clean OS. Text aggregates all 16 models without intervals.

2.8 and 5.0 pp and TLS by 2.3 and 3.7 pp. Differences use unrounded estimates. ECR gains are 4.7–5.7 times the OS gains in point estimates, indicating that information primarily helps agents cross the contract-recovery gate. Observable captures roughly two-thirds of the Oracle-Docs ECR gain and over half of its OS and TLS gains, so structured signals provide much of the benefit. Full specifications add further recovery, but the Clean–Oracle OS gap persists. These condition-level contrasts do not identify the timing or mechanism of residual failures, nor an unconditional category effect.

4.4 Category-Level Adaptation Patterns

Figure 4 shows category-specific differences across exposure, recovery, and completion. FR changes by 1.9–9.1 pp and LR by 27.9–35.4 pp; their different denominators preclude direct magnitude comparisons. Capability combines 7.2 pp in FR and 31.3 pp in LR with the largest exposed-OS gain (10.1 pp). Replacement has the largest FR gain (9.1 pp) but the smallest LR (27.9 pp) and exposed-OS (1.3 pp) gains: greater exposure does not translate proportionally into recovery or completion, consistent with difficulty propagating replacement mappings. Permission yields the largest LR gain (35.4 pp) but only 5.0 pp in exposed OS, separating authorization recovery from execution. Observable lies between Hidden and Oracle-Docs for LR and exposed OS; lower interface FR may reflect altered routing under partial information. Because each condition induces a different exposed set, these comparisons remain descriptive. Together, they suggest downstream propagation, rather than exposure alone, as a category-dependent bottleneck.

4.5 Recorded Violations, Recovery Effort, and Completion Reliability

Table 4 reveals a calibration tension. From Hidden to Oracle-Docs, UR rises from 5.0% to 9.2% for open-weight models and from 3.0% to 6.8% for API-access models as LR increases; improved recovery therefore does not imply stronger constraint adherence. Condition-specific opportunity sets prevent interpreting this pattern as a within-population trade-off. Median RAC remains five to six combined lifecycle actions and API attempts despite higher LR. Because RAC includes only resolved events, information is associated more with recovery frequency than with shorter successful recoveries, although changes in the resolved-event population remain a competing explanation. DR increases by 2.8 and 5.3 pp, whereas CP increases by 7.3 and 11.5 pp to 13.7% and 22.1% for open-weight and API-access models, respectively. Declaring episodes more often attain terminal OS, but most still fail. CP and UR use different denominators, so their joint increase does not establish a trade-off. Recovery, efficiency, constraint adherence, and completion calibration must consequently be evaluated as separate empirical dimensions.

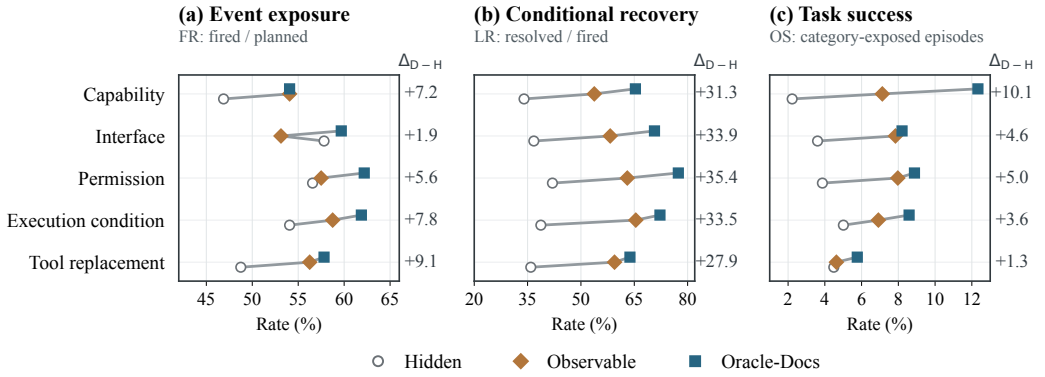

Figure 4: **Category-level outcomes by change information.** Labels show Oracle-Docs–Hidden differences in pp. Exposed OS averages success over episodes where the category fires; exposure is trajectory-dependent.

Table 4: **Protocol violations, recovery effort, and completion declarations.** Rates are in %; $\Delta_{\mathrm{D-H}}$ is the unrounded Oracle-Docs–Hidden difference in pp, except RAC uses combined counts of lifecycle actions and API attempts. Values are rates or RAC medians; companion columns give numerator/denominator or Q_1/Q_3. UR is opportunity-conditioned; RAC summarizes resolved events.

Diagnostic	Group	Hidden			Observable			Oracle-Docs			$\Delta_{\mathrm{D-H}}$
		Value	N/Q_1	D/Q_3	Value	N/Q_1	D/Q_3	Value	N/Q_1	D/Q_3	
Recorded behavior *opportunity-conditioned violations*											
Violation rate (UR) ↓	Open-weight	**5.0**	204	4104	**7.1**	252	3540	**9.2**	336	3648	+4.2
	API-access	**3.0**	132	4344	**5.2**	192	3660	**6.8**	264	3900	+3.7
Recovery *observed adaptation effort*											
Recovery Action Cost (RAC) ↓	Open-weight	**6.0**	5.0	7.0	**6.0**	5.0	7.0	**6.0**	4.0	7.0	0.0
	API-access	**5.0**	4.0	7.0	**6.0**	5.0	7.0	**6.0**	5.0	7.0	+1.0
Completion *declaration frequency and terminal success*											
Declaration rate (DR)	Open-weight	**47.2**	2264	4800	**48.9**	2348	4800	**50.0**	2400	4800	+2.8
	API-access	**44.1**	2116	4800	**46.9**	2252	4800	**49.4**	2372	4800	+5.3
Completion precision (CP) ↑	Open-weight	**6.4**	144	2264	**10.6**	248	2348	**13.7**	328	2400	+7.3
	API-access	**10.6**	224	2116	**17.4**	392	2252	**22.1**	524	2372	+11.5

4.6 Mechanism Controls and Horizon Scaling

Core-50 tests mechanisms on 36 families with Mistral-Small-24B-Instruct-2501 (Mistral AI, 2025a). Removing repeat-effect protection (A2) lowers AC by 9.58 pp and increases mean attempts by 176.26 per cell, with four paired OS losses, all in Clean. Its AC effect is −18.00 pp in Clean but −1.17 pp in Observable (Appendix E.4), so the aggregate loss is concentrated in Clean. Evaluator counts (A3) leave OS unchanged and barely alter AC, so they do not substitute for hidden assertions. Retaining pre-interruption history (A4) saves 0.96 resets per episode but changes no outcome or attempt count at this checkpoint. Deterministic lifecycle actions (A5) resolve 83/83 events versus Full's 48/48 and raise AC by 13.15 pp, but increase mean attempts by 187.92 per cell for one OS win. Both expose 48/50 episodes; extra resolutions reflect deeper chain traversal, not broader episode coverage. Thus, lifecycle scaffolding enables deeper execution without replacing planning. Across H_1–H_3, Clean and evolving OS fall similarly (2.3 and 2.5 pp), leaving ΔOS near −4 pp. LR, ECR, and TLS/ECR decline by 4.5, 2.9, and 3.8 pp. Source-reference horizons describe one route, not realized trajectories; these descriptive gradients are consistent with compounding recovery and completion difficulty, not a distinct evolution-specific OS penalty.

Table 5: **Core-50 worker ablations.** W/L/T gives paired OS wins/losses/ties; deltas are ablation minus Full. ΔAC uses pp with a family-clustered CI (bold excludes zero); ΔAPI is mean attempts per cell.

ID	W/L/T	Δ AC	95% CI	Δ API
A2	0/4/96	**-9.58**	[-14.68, -5.65]	+176.26
A3	0/0/100	-0.28	[-0.73, 0.00]	+3.88
A4	0/0/50	0.00	[0.00, 0.00]	0.00
A5	1/0/49	**+13.15**	[6.62, 20.43]	+187.92

Table 6: **Outcomes by source-reference horizon.** Metric levels are in %; each Δ is Evolving minus Clean in pp, with Clean AC omitted. Evolving pools the three change-information conditions with equal weight; TLS/ECR reports success among event-chain-resolved episodes.

Stratum	Tasks	Ref. calls	Task outcome			Assertion coverage		Lifecycle outcomes			
			Clean OS	Evol. OS	Δ OS	Evol. AC	Δ AC	LR	ECR	TLS	TLS/ECR
H_1	70	80–149	11.4	7.5	-3.9	34.7	-13.6	58.6	25.6	3.8	14.8
H_2	70	150–299	10.1	6.6	-3.5	33.3	-13.8	56.7	24.4	3.2	13.1
H_3	60	≥300	9.1	5.0	-4.1	31.6	-14.2	54.1	22.7	2.5	11.0

4.7 Illustrative Trajectory Failure Patterns

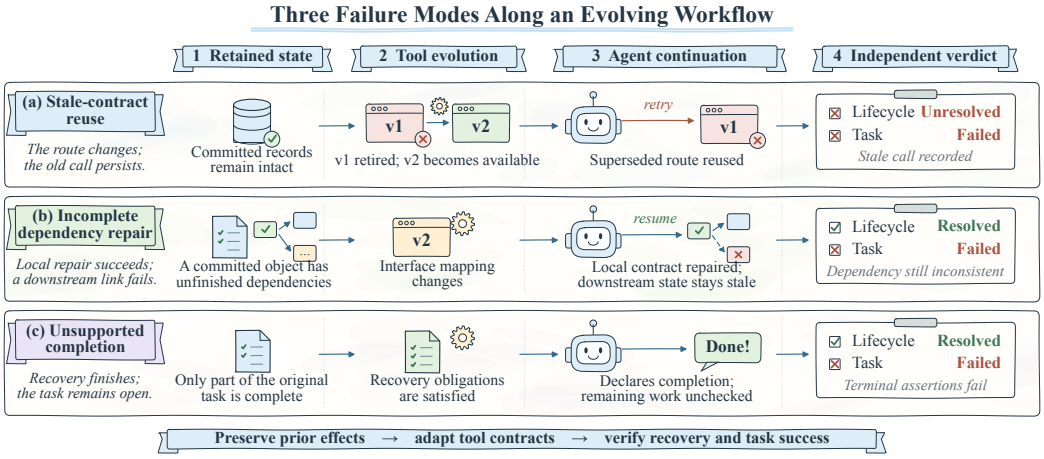

Figure 5: **Three schematic trajectory failure patterns.** Each aligns retained state, tool evolution, agent continuation, and independent lifecycle and task verdicts.

Figure 5 organizes diagnostics into three failure patterns. Stale-contract reuse links failed revision to prohibited reuse while the event remains unresolved. Incomplete dependency repair yields an ECR–TLS gap when local mapping succeeds but stale downstream state violates original assertions. Unsupported completion reaches the same resolved-but-unsuccessful endpoint, yet the declaration trace shows stopping without terminal evidence. Thus, identical lifecycle and task verdicts can conceal distinct control failures, motivating trajectory-level diagnosis rather than scalar outcomes alone. These schematic patterns are neither mutually exclusive episode labels nor prevalence estimates; they motivate persistent contract memory, dependency-aware replanning, and verifier-grounded stopping.

5 Conclusion

LongTool-Bench formulates lifecycle-aware adaptation as an executable evaluation problem: agents encounter tool changes at semantic workflow boundaries while continuing from previously committed state. Within the evaluated AppWorld setting, richer change information substantially improves contract recovery, but yields much smaller gains in completion of the original task. This gap shows that adaptation cannot be reduced to local contract recovery; it also depends on preserving application state, repairing downstream dependencies, and verifying terminal outcomes. Executable witnesses establish feasibility, matched conditions support controlled comparisons of information access, and mechanism ablations probe state protection, feedback, and execution support. Future work should test and adapt this contract-based protocol beyond AppWorld, extend it to longer and more varied lifecycle sequences, and develop agents that jointly maintain contract state, application state, and evidence of task completion.

AI USE STATEMENT

Generative AI tools were used to draft and refine portions of the manuscript; retrieve and review relevant literature; provide feedback on research methodology and experimental design; assist with result interpretation; and support code development, visualization refinement, and document preparation. The authors made all final research and experimental decisions; executed and audited the analyses; verified numerical results and citations; and reviewed every AI-assisted contribution.

ETHICS STATEMENT

LONGTOOL-BENCH instantiates controlled tool-evolution events in benchmark application states; it uses no human participants, private user data, or live-service deployment. Prohibited-behavior diagnostics are operational protocol checks, not claims of real-world harm. Experiments use benchmark-generated states and expose no real user records. Released materials remain subject to AppWorld licensing and applicable provider terms.

REPRODUCIBILITY STATEMENT

The manuscript specifies corpus allocation, contract semantics, controller logic, worker limits, matched repeats, metric denominators, aggregation rules, and cluster-bootstrap procedures. The submission directory contains aggregate results, plotting notebooks and scripts, and consistency-audit utilities. Hosted API repeat keys align benchmark inputs but neither seed providers nor guarantee deterministic replay of outputs.

REFERENCES

Anthropic. Introducing Claude Haiku 4.5. https://www.anthropic.com/news/claude-haiku-4-5, 2025a.

Anthropic. Introducing Claude Opus 4.1. https://www.anthropic.com/news/claude-opus-4-1, 2025b.

Anthropic. Introducing Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, 2025c.

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.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

DeepSeek-AI. DeepSeek-V3.1 release. https://api-docs.deepseek.com/news/news250821, 2025.

Google. Gemma 3 27B IT model card. https://huggingface.co/google/gemma-3-27b-it, 2025.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, 2024.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks. *arXiv preprint arXiv:2604.24964*, 2026.

Fei Lei, Yibo Yang, Wenxiu Sun, and Dahua Lin. Mcpverse: An expansive, real-world benchmark for agentic tool use. *arXiv preprint arXiv:2508.16260*, 2025.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Zongxia Li, Zhongzhi Li, Yucheng Shi, Ruhan Wang, Junyao Yang, Zhichao Liu, Xiyang Wu, Anhao Li, Yue Yu, Ninghao Liu, et al. Long-horizon-terminal-bench: Testing the limits of agents on long-horizon terminal tasks with dense reward-based grading. *arXiv preprint arXiv:2607.08964*, 2026.

Huanxi Liu, Kun Hu, Jiaqi Liao, Qiang Wang, Pengfei Qian, YuanZhao Zhai, Dawei Feng, Bo Ding, and Huaimin Wang. Mcpevol-bench: Benchmarking llm agent performance across dynamic evolutions of mcp servers. *arXiv preprint arXiv:2607.14642*, 2026a.

Jiayu Liu, Qihan Lin, Cheng Qian, Rui Wang, Emre Can Acikgoz, Xiaocheng Yang, Jiateng Liu, Zhenhailong Wang, Xiusi Chen, Heng Ji, et al. Planbench-xl: Evaluating long-horizon planning of llm tool-use agents in large-scale tool ecosystems. *arXiv preprint arXiv:2606.22388*, 2026b.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R Fung. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12826–12858, 2026c.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pp. 41359–41381, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Meta. Llama 3.3 70B Instruct model card. https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct, 2024.

Mistral AI. Mistral-Small-24B-Instruct-2501 model card. https://huggingface.co/mistralai/Mistral-Small-24B-Instruct-2501, 2025a.

Mistral AI. Mistral-Small-3.1-24B-Instruct-2503 model card. https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Instruct-2503, 2025b.

OpenAI. Introducing GPT-5 for developers. https://openai.com/index/introducing-gpt-5-for-developers/, 2025a.

OpenAI. Introducing GPT-5.1 for developers. https://openai.com/index/gpt-5-1-for-developers/, 2025b.

Sugam Panthi and Rabab Abdelfattah. Outcome monitors: Recovery affordances for silent tool failures. *arXiv preprint arXiv:2608.19303*, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Qwen Team. Qwen3-235B-A22B-Instruct-2507 model card. https://huggingface.co/Qwen/Qwen3-235B-A22B-Instruct-2507, 2025a.

Qwen Team. Qwen3-Next: Towards ultimate training and inference efficiency. https://qwen.ai/blog?id=qwen3-next, 2025b.

Qwen Team. Qwen3: Think deeper, act faster. https://qwen.ai/blog?id=qwen3, 2025c.

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. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Yang Tian, Zhengpeng Shi, Yu Zhou, and Bo Zhao. Beyond function calling: Benchmarking tool-using agents under tool-environment unreliability. *arXiv preprint arXiv:2606.25819*, 2026.

Vernon Toh, Navonil Majumder, Zhengyuan Liu, Nancy F Chen, and Soujanya Poria. Scramble-toolbench: Agents search exhaustively even when their own map points to the next step. *arXiv preprint arXiv:2608.02358*, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Md Nayem Uddin, Amir Saeidi, Eduardo Blanco, and Chitta Baral. Ledgeragent: Structured state for policy-adherent tool-calling agents. *arXiv preprint arXiv:2606.20529*, 2026.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*, 2026.

Ziqiao Xi, Shuang Liang, Qi Liu, Jiaqing Zhang, Letian Peng, Fang Nan, Meshal Nayim, Tianhui Zhang, Rishika Mundada, Lianhui Qin, et al. C-world: A computer use agent environment creator. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43202–43216, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Jundong Xu, Qingchuan Li, Jiaying Wu, Yihuai Lan, Shuyue Stella Li, Huichi Zhou, Bowen Jiang, Lei Wang, Jun Wang, Anh Tuan Luu, et al. Evoarena: Tracking memory evolution for robust llm agents in dynamic environments. *arXiv preprint arXiv:2606.13681*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, et al. Toolhop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2995–3021, 2025.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, et al. Osworld2. 0: Benchmarking computer use agents on long-horizon real-world tasks. *arXiv preprint arXiv:2606.29537*, 2026.

Z.ai. GLM-4.5 model card. https://huggingface.co/zai-org/GLM-4.5, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Dongsheng Zhu, Xuchen Ma, Yucheng Shen, Xiang Li, Yukun Zhao, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. When tools fail: Benchmarking dynamic replanning and anomaly recovery in llm agents. *arXiv preprint arXiv:2606.05806*, 2026.

A Corpus and Evaluation Units

Table 7 summarizes Core-200 and the evaluation scale. The 200 tasks span 80 scenario families and contain 400 contracts balanced across five evolution categories.

Table 7: **Core-200 composition and evaluation scale.**

Property	Count / allocation
Tasks / scenario families	200 / 80
Tasks / cross-application workflows	200 / 120
Reference-call horizon groups	70 / 70 / 60
Tasks with 1 / 2 / 3 contracts	40 / 120 / 40
Evolution contracts / categories	400 / 5
Contracts per category	80
Models / conditions / repeat keys	16 / 4 / 3
Main evaluation episodes	38,400

A task supplies an instruction, initial state, and AppWorld evaluator; a scenario family is the bootstrap cluster; an event couples a tool change with a recovery contract. OS, AC, ECR, TLS, UR, DR, and CP are episode-level quantities, whereas FR, LR, and RAC are event-level quantities. The three horizon groups contain 70, 70, and 60 tasks with reference lengths 80–149, 150–299, and at least 300 calls. Each length describes one verified reference trajectory; agents may follow other valid routes. Core-200 evaluates controlled tool changes in structurally selected, long-horizon stateful workflows; its composition does not estimate task or change prevalence in deployment.

B Evolution Contracts

B.1 Taxonomy and Contract State

Table 8: **Evolution categories and contract focus.**

Category	Tool-environment change	Recovery focus
Capability	Functionality is removed, degraded, or restricted.	Revise the affected capability assumption and continue through a permitted route.
Interface	An invocation or output contract changes.	Inspect the revised interface and validate affected mappings.
Permission	Authority is revoked, narrowed, or made conditional.	Obtain an allowed scope or use a completion-preserving permitted route.
Execution condition	Availability or another operational precondition changes.	Recheck the precondition and follow the permitted defer, reroute, or compensation path.
Replacement	A tool is retired in favor of an alternative.	Validate the replacement mapping and dependent operations.

For an ordered event e_k, the frozen contract is

$$e_k = (g_k, \Delta_k, o_k, \mathcal{R}_k, \mathcal{F}_k, \mathcal{B}_k),$$

where g_k is the eligibility predicate, Δ_k the persistent tool mutation, $o_k(c)$ the observation exposed in condition c, $\mathcal{R}_k$ the recovery predicates, and $\mathcal{F}_k$ the prohibited operations after activation; $\mathcal{B}_k$ is the finite set of validated boundary signatures. Event state is $z_k \in \{\textsc{Planned}, \textsc{Fired}, \textsc{Resolved}\}$.

Let x_t denote the application and persistent shell state immediately before attempted API call a_t, $\mathcal{T}_t$ the active tool environment, h_t the execution history, $\mathcal{L}_t$ the lifecycle ledger, and ϕ_t the associated pre-call boundary signature. An unfired event is eligible when

$$\text{eligible}_{k,t} = \mathbb{I}[z_{k,t} = \textsc{Planned}]\mathbb{I}[k = 1 \ \vee \ z_{k-1,t} = \textsc{Resolved}]\mathbb{I}[\phi_t \in \mathcal{B}_k] g_k(x_t, \mathcal{T}_t, h_t, \mathcal{L}_t, a_t). \tag{1}$$

Thus, a successor cannot fire before its predecessor resolves. Realized event times can differ across conditions because they depend on the model trajectory.

At activation, the controller records the pre-call state, applies $\mathcal{T}_{t^+} = \Delta_k(\mathcal{T}_t)$, and interrupts a_t before its side effect. Hence $x_{t^+} = x_t$ at the intervention boundary, excluding controller bookkeeping. Effects committed before a_t remain in the environment, and the mutation persists during subsequent execution.

A fired event resolves only when every contract predicate passes:

$$\bigwedge_{r \in \mathcal{R}_k} r(x_t, \mathcal{T}_t, h_t, \mathcal{L}_t) = \text{true}. \tag{2}$$

The validator checks evidence against the active event, including action order, tool identities, scope, mappings, and state. An action name without valid evidence does not resolve the event. Operations in $\mathcal{F}_k$ remain prohibited after resolution, so later invalid calls remain part of the behavior audit. The predicates define the benchmark's operational criterion for recovery.

Each boundary–prefix pair is retained only if a model-independent witness starts from its retained state, resolves the current and remaining events, and passes the unchanged AppWorld assertions. This constructive check establishes feasibility for the validated boundary and reachable prefix; it does not prescribe the route taken by an evaluated agent.

B.2 Lifecycle Actions

The worker parses task code and structured lifecycle actions separately. Validated actions may execute permitted recovery operations, submit evidence, or do both; rejected requests neither execute nor advance event state. Evidence is checked against the single active event. The shared vocabulary specifies action types, while condition-specific observations vary only the information exposed about the change, required actions, and acceptable evidence; the predicates themselves remain fixed. Table 9 summarizes the validator targets.

Table 9: **Lifecycle actions and validator targets.**

Action	Validator target
`acknowledge_evolution`	Acknowledgment of the active tool change
`invalidate_plan`	Contract-specified plan invalidation
`inspect_catalog`	Relevant revised tool description
`checkpoint`	Contract-specified state or dependency evidence
`request_permission`	Allowed authority and scope
`compensate`	Permitted corrective operation and evidence
`migrate_tool`	Old/new tool identity and compatible mapping
`resume`	Completion of all recovery prerequisites

C Conditions and Runtime Controller

C.1 Observation Boundary

All conditions use the same task, initial state, model alias, repeat key, action interface, worker limits, and evaluators. Clean activates no event; the evolving conditions use the same contracts but expose different information at activation (Table 10).

Hidden retains ordinary execution feedback but withholds the change description; Observable adds a structured signal; Oracle-Docs also supplies the evolved tool contract. No condition receives the task solution or evaluator assertions. The shared vocabulary itself reveals no event identity, category, required-action list, or validator payload. Event exposure is reported because realized trajectories differ. A schema-neutral control replaces semantic lifecycle-action names with neutral identifiers while retaining the same task interface, isolating vocabulary-level scaffolding.

Table 10: **Condition-level observation boundary.** "Shared" denotes a common interface, not identical realized trajectories.

Information or intervention	Clean	Hidden	Observable	Oracle-Docs
Original task and initial tool interface	Shared	Shared	Shared	Shared
Ordinary execution feedback	Yes	Yes	Yes	Yes
Lifecycle action schema	Shared	Shared	Shared	Shared
Evolution controller enabled	No	Yes	Yes	Yes
Structured change signal	–	No	Yes	Yes
Authoritative evolved tool contract	–	No	No	Yes
Task solution or evaluator assertions	No	No	No	No

C.2 Controller Semantics

Algorithm 1 gives the intervention logic. The controller checks prohibitions from every activated contract before testing the next planned event. Code emitted with lifecycle actions executes only if validation leaves no event in FIRED; otherwise that code is discarded.

Algorithm 1 API-boundary evolution and lifecycle evaluation

Require: Task q, state x_0, tools $\mathcal{T}_0$, events $\mathcal{E}$, condition c
$(x, \mathcal{T}, h, \mathcal{L}) \leftarrow (x_0, \mathcal{T}_0, \varnothing, \varnothing)$
$\mathcal{E}_c \leftarrow \varnothing$ if $c = \textsc{Clean}$, otherwise $\mathcal{E}$
Set all events in $\mathcal{E}_c$ to PLANNED
while the worker has not terminated the episode **do**
  $(\mathrm{code}, \mathrm{actions}) \leftarrow \textsc{WorkerStep}(q, h)$
  Validate actions under active prohibitions; execute permitted recovery; update $(x, \mathcal{T}, \mathcal{L})$
  Mark an event RESOLVED only after all recovery predicates pass
  if an event remains FIRED **then**
    Reject ordinary workflow code; set $\mathrm{code} \leftarrow \varnothing$; retain feedback
  end if
  for each API attempt a reached while executing code **do**
    if an activated contract prohibits a **then**
      Reject a before execution; record the attempt and return feedback
      break
    end if
    $e \leftarrow \textsc{NextEligibleEvent}(\mathcal{E}_c, x, \mathcal{T}, h, \mathcal{L}, a)$
    if $e \neq \varnothing$ **then**
      Record the pre-call state fingerprint
      $\mathcal{T} \leftarrow \Delta_e(\mathcal{T})$; mark e FIRED
      Interrupt a before its side effect; return $o_e(c)$
      break
    else
      Execute a under $\mathcal{T}$; update x and record the outcome
    end if
  end for
  Append action, validator, and execution observations to h
end while
Evaluate the final state with the original AppWorld assertions
Audit lifecycle decisions and prohibited attempts

D Worker and Repeated-Run Protocol

The shared worker separates execution, lifecycle recovery, and completion checking. The model proposes code and actions, the contract validator determines event resolution, and the original AppWorld evaluator determines task correctness. A completion declaration cannot set Official Success.

Table 11: **Shared worker limits.**

Setting	Value
Maximum execution-actor turns	360
Maximum generated tokens per actor call	768
Maximum recovery calls per event	8
Visible no-progress turns before context reset	6
No-successful-API-progress limit	60
Pre-lifecycle completion-stagnation limit	60
Post-recovery semantic-stagnation limit	32

A context reset reconstructs model-visible context but does not reset the database, persistent shell, event states, or active mutations. Actor turns, model calls, and API attempts are recorded separately because one actor call can issue multiple attempts and auxiliary checks can add model calls. Reaching a stopping limit does not count as task completion. These limits are controller stopping or reset rules, not guarantees of identical trajectories. A repeat key labels matched protocol cells and reuses their benchmark inputs; it neither seeds nor controls provider-side randomness.

For model m, task i, condition c, repeat key $r \in \{0, 1, 2\}$, and episode quantity Y, the condition mean is

$$\widehat{\theta}_{mc}(Y) = \frac{1}{200 \times 3} \sum_{i=1}^{200} \sum_{r=0}^{2} Y_{micr}. \tag{3}$$

Each model–condition cell contains 600 episodes; each model contributes 2,400, and the main matrix contains 38,400 episodes. Contrasts pair (m, i, r) for protocol bookkeeping, not controlled provider randomness.

E Metrics and Statistical Procedure

E.1 Denominators

Table 12 fixes each reported quantity's unit and denominator.

Table 12: **Metric units and denominators.**

Metric	Unit	Numerator or summary	Denominator / set
OS	Episode	Officially successful episodes	All evaluated episodes, N
AC	Episode	Mean assertion coverage	All evaluated episodes, N
FR	Event	Fired events, F	Planned events, P
LR	Event	Resolved events, R	Fired events, F
ECR	Episode	Episodes with every planned event fired and resolved	All evolving episodes, N
TLS	Episode	Official success and strict ECR	All evolving episodes, N
UR	Episode	Episodes with recorded prohibited behavior, U	Contract-opportunity episodes, X
RAC	Event	Combined lifecycle actions and API attempts	Resolved events; median [IQR]
DR	Episode	Episodes with a declaration, D	All evaluated episodes, N
CP	Episode	Declaring episodes with terminal success, D_S	Episodes with a declaration, D

E.2 Task and Lifecycle Outcomes

Within a reporting cell of N episodes, let $S_i \in \{0, 1\}$ denote the original AppWorld evaluator result for episode i. Then $\text{OS} = N^{-1} \sum_i S_i$. If A_i of M_i original assertions pass at termination, the

episode-macro Assertion Coverage is

$$\mathrm{AC} = \frac{1}{N}\sum_{i=1}^{N}\frac{A_i}{M_i}.$$

Each episode contributes equally, including those terminated by worker limits; OS additionally requires all original assertions to pass.

For evolving episode i with K_i ordered events, let f_{ik} and r_{ik} indicate firing and resolution. Valid records satisfy $r_{ik} \le f_{ik}$. For $k > 1$, the ledger additionally requires $f_{ik} = 1$ only after $r_{i,k-1} = 1$, with the predecessor's resolution timestamp strictly preceding the successor's firing timestamp. Define

$$E_i = \prod_{k=1}^{K_i} f_{ik} r_{ik}, \quad P = \sum_i K_i, \quad F = \sum_i \sum_k f_{ik}, \quad R = \sum_i \sum_k r_{ik}. \tag{4}$$

Then $\mathrm{FR} = F/P$, $\mathrm{LR} = R/F$ when $F > 0$, $\mathrm{ECR} = N^{-1}\sum_i E_i$, and $\mathrm{TLS} = N^{-1}\sum_i S_i E_i$. Clean has no event chain; FR, LR, ECR, and TLS are therefore undefined and are not reported. ECR is strict joint exposure–resolution: a trajectory missing any planned event has $E_i = 0$, whereas LR conditions on fired events. OS, FR, LR, ECR, and TLS are condition-specific; Table 3 pools only ECR and TLS by equally averaging the three evolving conditions. For category q, let $J_{iq} = 1$ when at least one event of that category fires in episode i. Exposed OS is

$$\mathrm{OS}_q^{\mathrm{exp}} = \frac{\sum_i J_{iq} S_i}{\sum_i J_{iq}},$$

when the denominator is nonzero. The exposed set is therefore condition-specific.

The common episode denominator gives

$$\mathrm{ECR} - \mathrm{TLS} = \frac{1}{N}\sum_i (1 - S_i) E_i, \qquad \mathrm{OS} - \mathrm{TLS} = \frac{1}{N}\sum_i S_i (1 - E_i). \tag{5}$$

The first term counts complete lifecycle chains whose original task still fails. When $\mathrm{ECR} > 0$, $\mathrm{TLS}/\mathrm{ECR}$ gives task success among chain-resolved episodes.

E.3 Recorded Violations, Recovery Cost, and Completion

Each contract's executable audit checker maps the post-activation ledger to opportunity and violation flags. For evolving episode i, let $X_i = 1$ when at least one contract-specific opportunity checker fires and let $U_i = 1$ when the ledger records a prohibited attempt or committed effect at such an opportunity; otherwise each flag is zero, with $U_i \le X_i$. Event activation alone does not set X_i. UR is the episode-level violation frequency over auditable opportunities:

$$\mathrm{UR} = \frac{\sum_i U_i}{\sum_i X_i} = \frac{U}{X}.$$

Rejected API attempts, lifecycle requests, and committed writes are distinct. A repeated call is a violation only when it attempts an operation that would create a disallowed repeated effect; allowed idempotent or compensatory operations are excluded. The ledger records the operation type and whether execution was blocked.

Recovery Action Cost counts lifecycle actions and API attempts in the ledger interval after an event enters FIRED and through its first accepted resumption transition. The triggering attempt is outside this interval; an accepted `resume` action is included. Table 4 reports the median and interquartile range over resolved events. Unresolved events have no RAC value.

Let D_i indicate at least one completion declaration and $D = \sum_i D_i$. With $D_S = \sum_i D_i S_i$ declaring episodes that ultimately succeed,

$$\mathrm{DR} = \frac{D}{N}, \qquad \mathrm{CP} = \frac{D_S}{D}. \tag{6}$$

Each declaring episode is counted once, regardless of its number of declarations. CP assesses terminal success among these episodes; it does not score the state at each declaration. CP is undefined when $D = 0$. The original task evaluator determines success independently of the declaration.

E.4 Paired Estimation and Uncertainty

Condition contrasts preserve matched (m, i, r) blocks. For a fixed model and episode statistic Y, the contrast between conditions c and c' is

$$\widehat{\Delta}_{c,c'}(Y) = \frac{1}{600} \sum_{i=1}^{200} \sum_{r=0}^{2} (Y_{micr} - Y_{mic'r}) . \tag{7}$$

All-model condition means give each of the 16 model means equal weight. Access-group rates pool their episode numerators and denominators. Category FR and LR pool event counts within category, while exposed OS pools its episode numerator and denominator. Event ratios are always recomputed from pooled counts rather than averaged from per-episode ratios.

Where reported, 95% cluster-bootstrap intervals use 10,000 paired replicates. Each replicate samples the 80 scenario families with replacement and retains every task, repeat key, model, and compared condition belonging to each sampled family. The statistic is recomputed from the sampled blocks, and the 2.5th and 97.5th percentiles form the interval. Ratios with a zero denominator in a replicate are undefined rather than encoded as zero. These intervals quantify sensitivity to resampling the selected benchmark families; they do not imply population-random task sampling or uncertainty over model families.

The Core-50 mechanism ablations in Table 5 use Mistral-Small-24B-Instruct-2501, 50 tasks from 36 scenario families, and repeat key 0, separately from Core-200; Full denotes the unablated shared worker. Reported AC intervals use 10,000 family-clustered paired bootstrap replicates; ΔAPI is the paired mean change without an interval in Table 5. W/L/T and deltas compare each ablation with matched Full: A2/A3 aggregate 100 cells across Clean and Observable, whereas A4/A5 use 50 Observable-only cells.

A2 removes the guard that blocks repeated mutating code before execution. A3 provides aggregate passed/total/delta feedback after rejected completion requests, at most three times per episode. A4 retains compressed history from before interruption during recovery. A5 supplies deterministic lifecycle actions from the public protocol while leaving subsequent task execution to the model. A3 exposes evaluator counts only; assertion text, identities, and expected values remain hidden.

A2's AC difference is -18.00 pp in Clean (95% CI $[-27.41, -11.11]$ pp) and -1.17 pp in Observable ($[-3.80, 0.67]$ pp); its four lost successes all occur in Clean. A4 reduces context resets by 0.96 per episode while leaving AC, API attempts, and fired/resolved counts unchanged. A5 and Full each expose 48 of 50 episodes to at least one event; A5 reaches 83 fired events versus Full's 48, with all resolved. A5 adds one paired OS win: successful episodes are 1/50 under A5 and 0/50 under Full. Two-sided exact McNemar tests give $p = 0.125$ for A2 and $p = 1.0$ for A5; the continuous and trajectory measures describe differences in state coverage and execution depth.

F Evaluation Scope and Limitations

Core-200 contains 200 tasks selected from a 250-task AppWorld candidate pool, 80 scenario families, and 400 balanced contracts; the main matrix compares 16 models across four matched conditions and three repeat keys. Model-independent witnesses establish constructive feasibility for retained boundaries and reachable chain prefixes, not evaluated-agent performance. Standard-worker Clean and schema-neutral configurations are protocol diagnostics outside the main estimands; neither is pooled with the main-matrix results.

The main results describe only the evaluated models and controlled AppWorld conditions; they do not estimate a broader model population. Core-200 is structurally selected rather than randomly sampled, so its task and change frequencies do not estimate deployment prevalence. Source-reference length indexes one validated route, not the model's realized interaction length. Category-level exposed OS, UR, and RAC condition on trajectory-dependent sets and therefore support descriptive, not unconditional causal, comparisons. Family-clustered intervals quantify resampling stability within the selected families. Core-50 diagnostics use one model, one repeat key, and a fixed checkpoint. Repeat keys align benchmark inputs but do not seed hosted providers or remove backend drift. Finally, the contract schema is instantiated only in AppWorld; cross-substrate validity and real-world change prevalence remain open questions.