

Why AI Agents Fail at Long-Horizon Tasks

A research perspective on state tracking, cascading errors, verification, and the environments needed to evaluate sustained autonomous work

Deaimer Research Team

Abstract

AI agents can complete impressive isolated actions while struggling to finish extended workflows. The gap is not adequately explained by context length or raw reasoning ability alone. Long-horizon tasks require agents to maintain a coherent task state, coordinate dependencies, respond to changes, decide when to seek evidence, and verify the consequences of actions. This article synthesizes primary research on interactive benchmarks, computer-use environments, and long-horizon collaboration, with particular attention to the 2026 OSWorld 2.0 release. It distinguishes documented findings from a proposed evaluation framework for studying error propagation and recovery. The central argument is that dependable agents need both better control of their own work and training and evaluation environments that expose the costs of losing that control. No new Deaimer experiment or performance result is claimed.

1. Introduction: the unfinished task problem

An AI agent is asked to prepare a complex reimbursement request. It must consult a policy, locate receipts, reconcile inconsistent records, complete a form, attach evidence, and confirm that the submitted claim is correct. Each individual operation may be straightforward. The overall workflow is not. An incorrect amount copied early can contaminate later fields. A newly arrived message may invalidate a previous assumption. A final confirmation screen may be mistaken for successful submission. This is an illustrative example, not a reported Deaimer test.

Long-horizon reliability is the ability to maintain acceptable behavior and reach the intended outcome across a sequence of interdependent decisions. It is distinct from performing many actions. An agent can take hundreds of steps without progressing toward a coherent objective. Conversely, a short sequence can be demanding if later decisions depend on information discovered earlier. Horizon should therefore be understood through dependencies, uncertainty, changing state, and verification requirements rather than tool-call count alone.

Research benchmarks have made these challenges observable. AgentBench established a framework for evaluating language-model agents in interactive environments [1]. WebArena offered reproducible web workflows and programmatic success checks [2]. OSWorld brought execution-based assessment into desktop applications [3]. These are important foundations, not obsolete systems. More recently, OSWorld 2.0 deliberately expanded the task horizon and the complexity of real computer workflows [4]. The progression illustrates a shift from whether an agent can act to whether it can sustain coordinated work.

This article asks why extended tasks break down, how researchers can separate competing explanations, and what evidence should guide model development. Its proposed framework is an interpretation of the

literature, not a validated taxonomy or proprietary Deaimer invention. The practical goal is to replace vague claims about autonomous competence with testable questions about what an agent observes, remembers, changes, and checks.

2. What the latest long-horizon evidence shows

The OSWorld 2.0 paper, released in June 2026, reports 108 long-horizon workflows with a median skilled-human operation time of roughly 1.6 hours [4]. The authors describe tasks involving multiple applications, authentic artifacts, changing information, implicit state, and user clarification. Their benchmark is intentionally more demanding than the original OSWorld task distribution, and its results should not be confused with results on OSWorld 1.0 or OSWorld-Verified.

In the authors' reported 500-step evaluation, their strongest tested Claude Opus 4.8 configuration completed 20.6% of tasks under strict binary scoring while earning a 54.8% partial score [4]. These are measurements from the authors' specific setup and release, not a general ranking of every available agent. The difference between full and partial scores is instructive: meaningful intermediate work does not guarantee a correct finished result. An agent can achieve many local checkpoints while missing one dependency that determines overall success.

The OSWorld 2.0 authors attribute important failures to dropped constraints, missed updates, insufficient clarification, hidden-state reasoning, and skipped verification [4]. Their benchmark also includes safety auditing and fine-grained checks. It would be inaccurate to criticize the new release as though it still used only the original short, binary desktop tasks. The research value lies precisely in its updated design and in the distinction between progress and completion.

AgentWorld, released in September 2026, studies a different but related setting: sustained collaboration among multiple language-model agents in a game-based sandbox [5]. It uses tasks with more than 50 interaction rounds and investigates communication, shared planning, and causal contributions to group outcomes. Its findings cannot be directly generalized to enterprise desktop work, but they highlight a related failure mechanism: local competence does not automatically produce coordinated success when relevant state is distributed across participants.

These studies motivate a narrower conclusion than the slogan that agents cannot handle long tasks. They show that certain evaluated systems struggle under particular long-horizon conditions, and they supply ways to study those failures. New model releases, tool scaffolds, and task designs may change the measured gap. Reliable comparisons must identify the benchmark version, agent configuration, tool budget, and evaluation date.

3. Failure mechanism one: losing the task-level state

A long task creates several simultaneous forms of state. There is environment state: files, messages, application settings, database records, and the effects of previous actions. There is task state: the objective, requirements, completed subtasks, unresolved dependencies, and constraints. There is epistemic state: what the agent knows, what it suspects, and which claims still need confirmation. Confusing these layers can cause an agent to act confidently on an outdated premise.

Consider an illustrative coding workflow. The agent finds a dependency conflict and records that version A is installed. A repair command upgrades the dependency to version B, but the agent continues planning as though version A were present. A later failure is then interpreted using a stale explanation. The model may have enough context to read the update, yet still fail to revise its working state. The issue is not necessarily lack of storage. It is failure to maintain the relationship between observation, intervention, and current belief.

Longer context windows may reduce information loss, but they do not by themselves establish that the right facts will be selected, reconciled, and acted upon. A useful task-state representation should distinguish verified observations from hypotheses, attach time or version information to mutable facts, and mark assumptions that must be rechecked before consequential actions. This is a proposed design principle, not evidence that a particular memory format always improves outcomes.

A diagnostic experiment could vary only the availability and quality of state summaries while keeping the model, tasks, and tool access constant. Researchers could measure whether failures arise from missing information, incorrect retrieval, or inappropriate use of available information. Those outcomes suggest different interventions. Adding context may help the first; provenance and revision procedures may matter more for the others.

4. Failure mechanism two: small errors become dependencies

In a one-step question, a wrong answer can be isolated. In an extended workflow, an early error may become an input to later actions. A mistaken file path can cause an agent to modify the wrong configuration. A mistaken customer identifier can propagate across several documents. An incomplete test can make a broken repair look finished. Error propagation is therefore a structural property of a workflow, not merely a higher count of independent mistakes.

A simple illustrative model helps explain the intuition. If a workflow has n independent required steps, each with success probability p , then the probability that all succeed is p raised to n . At $p = 0.98$ and $n = 30$, this illustrative calculation gives roughly 0.545. Real agent actions are rarely independent, and recovery can reverse earlier mistakes; the formula is not a performance estimate. It shows why apparently strong local accuracy may be insufficient when every required condition must hold.

Dependencies make the problem more interesting. Some errors are recoverable, some are harmless, and some destroy information needed later. A sensible evaluation should record not only whether an error occurred but also whether it was detected, whether its effects spread, and whether the agent restored the intended state. An agent that recognizes a bad intermediate result and corrects it may be more useful than one that follows a flawless-looking but fragile path on a narrow test.

One research direction is to introduce controlled faults at specific points in otherwise equivalent tasks. The intervention could alter a tool response, invalidate an intermediate artifact, or introduce a conflicting instruction from an authorized user. By comparing recovery trajectories across conditions, researchers can estimate which failures stem from observation, planning, execution, or verification. Such studies must keep destructive and sensitive operations inside safe, resettable environments.

5. Failure mechanism three: planning without revision

Plans are useful because they organize dependencies. They become liabilities when an agent treats them as instructions that must be followed despite contrary evidence. Real workflows change: a build fails, a website changes its form, a collaborator supplies new information, or a file has already been modified. The correct next action depends on the current environment, not just on the original plan.

A robust agent needs at least two planning modes. Before acting, it should identify major subgoals, dependencies, constraints, and possible stopping conditions. During execution, it should compare observed results with expected results and revise the plan when the mismatch is material. This resembles closed-loop control: actions are selected using feedback rather than committed as an unchanging sequence. The analogy is useful but does not imply that every agent must implement a formal control-theoretic planner.

The OSWorld 2.0 design includes dynamic environment updates, including messages that arrive while a task is underway [4]. This is a meaningful improvement over tests where all relevant information is fixed at the beginning. It creates a measurable distinction between an agent that can follow a prepared script and one that can incorporate new evidence without losing track of the objective.

A practical test could hold the initial task constant and vary when an update arrives: before planning, after the first major action, or just before submission. Outcomes should include whether the update was noticed, whether affected decisions were revisited, and whether the final result complied with the revised requirements. The key is to evaluate appropriate revision, not to reward unnecessary replanning on every step.

6. Failure mechanism four: weak verification and premature completion

A task is not finished because an agent says it is finished. Completion should be grounded in the resulting environment state. A generated report may exist but contain the wrong figures. A repaired service may start once but fail after restart. An email may be drafted but never sent. The difference between plausible completion and verified completion is especially consequential when later users rely on the result.

WebArena and OSWorld advanced the field by evaluating actions against executable states rather than treating persuasive textual explanations as proof [2,3]. OSWorld 2.0 extends this approach with fine-grained checkpoints and audits designed to catch reward hacking and false negatives [4]. Its authors explicitly distinguish partial progress from complete workflow success. The benchmark does not eliminate all grading ambiguity, but its methodology addresses several problems that earlier, simpler evaluation designs could miss.

Verification should be risk-sensitive. Low-cost reversible actions may need lightweight checks. Destructive edits, submissions, payments, and changes to shared resources may require stronger confirmation and authorization. A uniform requirement to verify every trivial step could waste resources or make agents unusably slow. The research question is how to allocate verification effort where it changes expected outcomes.

A useful experimental comparison would test three policies: no explicit verification stage, verification only at final submission, and targeted checks at dependency boundaries. Researchers should report task completion, constraint violations, cost, latency, and error recovery. If verification improves correctness but doubles the interaction budget, that tradeoff matters. A credible result should not hide it inside a single success score.

7. Failure mechanism five: uncertainty and clarification

Long workflows often contain information that cannot be inferred safely from available artifacts. A user may have omitted a preferred account, a policy may allow multiple interpretations, or two records may disagree. An agent that always guesses may appear decisive while producing avoidable errors. An agent that always asks may be accurate but impose unacceptable interruption costs. Reliable behavior requires choosing when to inspect, ask, defer, or proceed.

OSWorld 2.0 includes tasks with a simulated user who can answer bounded clarification questions, and it evaluates proactive interaction as one of several challenge phenomena [4]. That feature matters because clarification is not merely a conversational courtesy. It is an action that can reduce uncertainty and alter what subsequent steps are authorized or sensible.

An illustrative procurement task may require choosing between two similarly named suppliers. If the records contain no decisive identifier, an agent should not invent one. It could inspect a purchase order, compare registration details, or ask an authorized person to resolve the ambiguity. Which action is best depends on information availability, urgency, and the cost of a wrong choice. A well-designed environment should make those consequences explicit.

For evaluation, researchers can construct matched tasks with sufficient evidence, missing evidence, and conflicting evidence. They can measure correct autonomous completion, unnecessary questions, unsupported guesses, and harmful actions. The desired behavior is calibrated intervention: neither blind confidence nor blanket refusal. This framing also helps enterprise buyers distinguish an agent that works independently from one that operates responsibly.

8. Failure mechanism six: tool and environment boundaries

An agent can reason correctly and still fail because the tool interface is misleading or the environment behaves unexpectedly. Web interfaces contain modal dialogs, stale elements, asynchronous updates, and permission boundaries. Terminal environments contain dependency changes, partial command failures, and persistent state. Cross-application tasks add serialization problems: information copied from one tool must retain its meaning in another.

The original OSWorld paper addressed real computer environments rather than abstract textual puzzles [3]. OSWorld 2.0 broadens that design through self-hosted services, authentic artifacts, dynamic updates, and more complex workflows [4]. These releases should be treated as distinct measurement instruments. A historical limitation of an early benchmark does not automatically apply to a later version that explicitly addresses it.

When diagnosing failure, researchers should separate model decisions from tool execution. Was the correct action selected but rejected by an interface? Did the tool return ambiguous feedback? Did the agent misread a successful action as a failure and repeat it? Did it lack authorization for a necessary step? These distinctions matter because a model-training intervention may not fix a broken tool contract.

A useful evaluation record should preserve tool inputs, tool outputs, state snapshots where feasible, and the agent's declared goal at important transitions. This does not require publishing confidential customer data. Synthetic or sanitized environments can preserve the causal structure of a workflow while avoiding exposure of private records. Trace inspection should focus on failure mechanisms rather than collecting every detail without purpose.

9. Measuring long-horizon reliability without misleading scores

Binary task success remains important: a user ultimately needs a completed outcome. Yet a single pass rate cannot explain why a workflow failed. Partial-credit metrics can distinguish progress from non-progress, but they can also reward shortcuts if checkpoints are poorly specified. The most useful evaluation suite therefore reports multiple dimensions rather than assuming that one scalar captures all operational value.

At minimum, consider final-state correctness, constraint compliance, repeated-run consistency, recovery from injected errors, clarification quality, and resource cost. Researchers should define which violations are unacceptable even when the primary objective is achieved. They should also record the version of the benchmark, agent scaffold, model, tools, and time or step limits. Without those controls, a claimed improvement may reflect a larger budget or a different harness.

A critical distinction is between action count and effective horizon. A task with fifty repeated clicks may be long in steps but simple in reasoning. A task with ten causally dependent decisions may be harder. An informative dataset should tag dependency depth, state volatility, number of sources, ambiguity, reversibility, and required verification. Those dimensions can then be tested against measured outcomes instead of being assumed to predict difficulty.

For statistical reporting, repeated trials and uncertainty intervals matter because model sampling and tool timing can change outcomes. If two systems differ by a small amount on a small task sample, the evidence may be inconclusive. For long tasks, evaluating many repeated runs can be expensive; researchers should report the actual sampling design rather than disguising sparse evidence with precise-looking percentages.

10. Proposed experiment: where does the failure curve bend?

The following is a proposed research protocol, not a completed experiment. Begin with a family of executable workflows that share an underlying objective but differ in dependency depth. Construct short, medium, and long variants by adding genuinely dependent subtasks, not irrelevant repetitions. Keep task wording, tools, and outcome checks as comparable as possible. Use synthetic records and controlled services to preserve privacy and resetability.

The first research question is whether completion decreases mainly with step count or with dependency complexity. The second asks whether explicit task-state tracking reduces errors that arise from stale

assumptions. The third asks whether targeted verification reduces the spread of errors introduced early in the workflow. A fourth asks whether the benefit of clarification increases when critical information is intentionally withheld.

Compare a baseline agent configuration with controlled variants that add a structured task ledger, dependency-boundary verification, or an evidence-seeking policy. Do not treat these additions as universally effective; each introduces compute, latency, and potential new failure modes. Hold the underlying model fixed when testing scaffolding effects, and separately test model changes if that is a research objective.

The evaluation should include held-out task families, new tool layouts, changed artifact formats, and dynamic updates. Report full completion, partial progress, prohibited effects, error detection, recovery, question frequency, and total cost. Review trajectories to distinguish the intervention's intended effect from shortcuts or evaluator exploitation. Predefine exclusion rules for invalid tasks and publish enough environment details to make the results interpretable.

Several outcomes would be scientifically useful. If structured state tracking helps only when tasks contain changing records, that suggests a targeted intervention. If verification improves safety but not completion, it may still be valuable for high-risk workflows. If gains vanish on held-out mechanisms, the system may have learned a template rather than a transferable skill. Null results would also matter because they would constrain claims about popular agent-design techniques.

11. Implications for model developers and enterprise buyers

Model developers should resist treating longer reasoning traces as a substitute for task-level evidence. More inference tokens may support better decisions, but they do not guarantee that the agent notices changed state, asks for missing information, or checks the final artifact. Research priorities should be based on observed failure modes. An agent that repeatedly forgets constraints needs a different intervention from one that cannot use a particular application interface.

Training environments should represent causal variation, not only superficial diversity. A collection of tasks that changes filenames while preserving one hidden solution pattern may overstate generalization. Better coverage varies failure causes, dependency structures, state representations, permissions, and the consequences of actions. Executable environments make these differences observable and can provide feedback that a static instruction-response dataset cannot.

Enterprise buyers should ask whether the evidence matches their workflow. A benchmark score on desktop navigation may be relevant but insufficient for a regulated approval process. Buyers can request a small pilot using representative, sanitized tasks and clear acceptance criteria. Useful questions include whether the agent can recover after an error, preserve existing data, handle ambiguity, and produce verifiable artifacts. The appropriate evaluation should reflect the cost of wrong actions as well as the value of completed work.

Procurement should also examine reproducibility and ownership: Can tasks be reset? Are grading rules inspectable? Are reference solutions and negative controls available? What software versions are pinned? Which actions are allowed? Does the supplier document task provenance and licensing? These questions

help distinguish a research-grade environment from a collection of demonstrations that merely look convincing.

12. Deaimer's relevance: environments for the full workflow

Deaimer's published research describes executable task environments and custom terminal-style datasets, including the Termetheus offering, as ways to study and develop agent capabilities through consequential interaction [6,7]. That public positioning connects naturally to long-horizon research. A useful task package can make state changes, dependencies, recovery opportunities, and final outcomes measurable. The connection is about the kinds of experiments such environments can support, not proof that any particular Deaimer collection already improves long-horizon performance.

For teams investigating sustained coding or operational work, a sensible starting point is a narrowly defined capability gap. Examples might include recovering from an incorrect dependency change, maintaining consistency across multiple files, or verifying that a repaired service remains healthy after restart. A controlled pilot can determine whether the environment represents the desired failure mechanism and whether the grader distinguishes legitimate success from shortcuts.

The larger lesson is independent of a vendor. Public benchmarks establish common reference points. Purpose-built environments allow controlled learning and diagnosis. Separate held-out evaluations test whether progress transfers. These functions complement one another, and claims become more credible when the evidence from each is kept distinct.

Teams exploring custom long-horizon agent tasks can review Deaimer's public environment work and discuss a scoped evaluation pilot. Any performance claim should follow controlled measurement on the target configuration, not precede it.

Conclusion: reliability is sustained control

Long-horizon agent failure is not one problem with one remedy. It is a family of failures involving state, dependencies, planning, uncertainty, tool execution, and verification. The research literature increasingly provides benchmarks that expose these mechanisms in realistic interactive settings. OSWorld 2.0 in particular demonstrates why progress on shorter computer-use tasks should not be mistaken for complete end-to-end workflow reliability [4].

The practical response is to evaluate agents as systems that must maintain control over changing work. Researchers should test which interventions improve transfer rather than assume that more steps, more context, or more data will suffice. Developers should inspect trajectories and measure recovery, not only final scores. Buyers should demand evidence that reflects their own task dependencies and operational constraints.

A capable agent is not merely one that knows the next action. It is one that can tell whether that action was appropriate, understand how it changed the task, revise its plan when reality disagrees, and verify that the work is genuinely complete.

References

- [1] Liu, X., et al. AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688 (2023, subsequent revisions). <https://arxiv.org/abs/2308.03688>
- [2] Zhou, S., et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. arXiv:2307.13854 (2023; ICLR 2024). <https://arxiv.org/abs/2307.13854>
- [3] Xie, T., et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. arXiv:2404.07972 (2024). <https://arxiv.org/abs/2404.07972>
- [4] Yuan, M., et al. OSWorld 2.0: Benchmarking Computer Use Agents on Long-Horizon Real-World Tasks. arXiv:2606.29537 (June 2026). <https://arxiv.org/abs/2606.29537>
- [5] Shu, R., et al. AgentWorld: Benchmarking Long-Horizon Collaboration of Multi-agent LLMs. arXiv:2609.31590 (September 2026). <https://arxiv.org/abs/2609.31590>
- [6] Deaimer. The Experience Layer of AGI: How Real-World Tasks Shape More General AI. October 12, 2026. <https://www.deaimer.com/research/the-experience-layer-of-agi-how-real-world-tasks-shape-more--02jnq>
- [7] Deaimer. Termetheus and the Foundations of Environment Based AI Learning. October 7, 2026. <https://www.deaimer.com/research/termetheus-and-the-foundations-of-environment-based-ai-learn-wwqsr>