

Beyond Benchmark Scores: Why Realistic RL Environments Matter for Reliable AI Agents

How executable tasks, verifiable feedback, and independent evaluation can connect leaderboard progress to transferable agent capability

By Deaimer Research Team

Abstract

Public agent benchmarks make progress visible, but a score on a fixed task distribution cannot by itself establish reliability in unfamiliar workflows. This research perspective explains how realistic, resettable reinforcement-learning environments complement public benchmarks by supporting consequential interaction, verifiable feedback, controlled experimentation, and capability-focused training. Drawing on AgentBench, WebArena, OSWorld, SWE-bench Verified, and published benchmark-maintenance evidence, it distinguishes task completion from safe and durable behavior, examines reward design and distribution shift, and proposes a falsifiable experiment on environment diversity and transfer. No new model-performance results are reported.

Introduction: why AI agent benchmarks are not the whole story

AI agent benchmarks provide measurements taken under particular conditions. A reliable AI agent is a system that can repeatedly reach useful outcomes when those conditions change. Confusing the two leads to a familiar research mistake: treating improvement on a fixed test as proof of transferable competence. The distinction matters especially for agents, because an agent does not merely answer a question. It observes a state, chooses an action, changes its environment, receives feedback, and decides what to do next. Every step can introduce new uncertainty.

Consider an illustrative coding agent asked to repair a failing service. A benchmark might check whether a prescribed test suite passes. A deployment owner might also care whether the agent preserved customer data, avoided exposing secrets, left the service restartable, and recorded enough evidence to explain its changes. Both views are legitimate, but they measure different things. The research question is not whether benchmarks should disappear. It is how to connect benchmarks to learning environments and evaluation procedures that measure the behavior a team actually wants.

This article develops a research perspective, not a new experimental result. It draws on interactive agent benchmarks, computer-use environments, software engineering evaluations, and published discussion of benchmark quality. Its central claim is conditional: realistic, resettable, verifiable reinforcement-learning environments can provide a useful complement to static leaderboards when their tasks, feedback, and held-out evaluations are designed to reflect target capabilities. Realism alone does not guarantee transfer. The burden is to demonstrate it.

1. What an agent benchmark actually measures

A language-model benchmark commonly maps a prompt to a scored response. An interactive agent benchmark evaluates a trajectory: a sequence of observations, actions, and resulting states. AgentBench helped establish this broader framing by evaluating language-model agents across multiple interactive environments and analyzing difficulties with longer-term reasoning, decisions, and instruction following [1]. Its contribution is methodological: it makes the agent-environment loop part of the object of study.

WebArena subsequently offered self-hosted web applications and programmatic success checks for realistic browsing tasks [2]. OSWorld extended the perspective to computer tasks across operating systems, with initial-state configurations and execution-based evaluators [3]. These systems differ in modality and scope, yet share a principle:

success must be tested against consequences, not merely plausible text.

A benchmark result is therefore a property of a configuration. That configuration includes the model, prompts, agent scaffold, tools, permitted actions, resource limits, environment version, task sample, and scoring rules. Two systems using the same underlying model can differ because one has a stronger browser interface or better error recovery. A ranking that omits these details can encourage overly broad conclusions.

The task distribution matters equally. A benchmark covering web forms may offer excellent evidence about navigation while saying little about incident response. A repository repair benchmark may reveal patching ability without fully characterizing operational safety. Neither limitation invalidates the benchmark. It defines the boundary of what its score supports.

This suggests three separate questions for any evaluation: Did the agent complete the sampled tasks? Did it complete them through acceptable behavior? Would the result persist on genuinely new tasks, states, and workflows? The first is often easiest to quantify. The second and third require richer environments and deliberate test design.

2. How reinforcement learning environments change AI agent training

In reinforcement learning, an agent's policy maps observations or histories to actions. An environment supplies state transitions and feedback. For a simplified episodic objective, researchers may optimize expected return, $J(\pi) = E[\text{sum from } t=0 \text{ to } T \text{ of } \gamma^t r_t]$, where r_t is reward, γ discounts later outcomes, and T is the episode horizon. This notation is familiar, but the hard design work is hidden inside the definition of state, actions, reward, and episode termination.

For AI agent training, an executable RL environment makes those choices concrete. A coding agent may need to inspect a repository, reproduce a bug, modify source, run tests, and interpret failures. A computer-use agent may navigate menus, recover from modal dialogs, and save artifacts in the correct location. A scientific agent may need to respect units, provenance, numerical tolerances, and domain-specific assumptions. In each case, the result depends on interactions across time rather than a single final string.

Realism is not identical to visual fidelity. A beautifully rendered interface with deterministic toy behavior may teach less about recovery than a modest command-line environment with real dependencies, noisy logs, and irreversible state changes. Conversely, an excessively complex environment can make learning inefficient or grading unreliable. The useful question is whether the environment preserves the causal structure of the target work: what information is available, what actions are legal, what can go wrong, and what constitutes success.

A well-designed task also needs a reset mechanism. Without repeatable initial states, changes in outcomes may reflect leftover files, caches, network services, or user data rather than learning. Resetability allows repeated trials and controlled comparisons. It is an experimental control, not merely an operational convenience.

Feedback quality is another central variable. Sparse final success signals are straightforward to interpret but may offer little guidance during long tasks. Intermediate signals can improve credit assignment, yet they can also encourage reward hacking if they privilege superficial milestones. A reward for 'tests passed' can reward disabling tests unless the verifier checks the integrity of the test suite. Reward design therefore needs both positive objectives and constraints on unacceptable shortcuts.

3. AI agent evaluation: the strengths and limits of public benchmarks

Public benchmarks provide a common language for progress. They allow independent groups to reproduce tasks, compare systems, investigate errors, and challenge evaluation assumptions. SWE-bench brought real repository issues into software-agent evaluation; SWE-bench Verified subsequently introduced expert review to address defects in parts of the original task set [4]. These are substantive contributions, not examples of a failed approach.

The same history shows why benchmarks require maintenance. In February 2026, OpenAI reported that SWE-bench Verified was no longer an adequate primary measure of frontier coding capability, citing contamination and remaining task-quality issues, and recommended SWE-bench Pro [5]. This is a time-specific assessment of that benchmark, not a claim that all software-agent benchmarks are contaminated. It also illustrates a productive scientific pattern: evaluate the evaluator, publish limitations, and revise the instrument.

OSWorld's original 2024 paper reported a large gap between its evaluated agents and human task completion [3]. Those numbers describe the models and setup of that release, not current agent capability. Repeating historical baseline percentages as though they were current rankings would mislead readers. Similarly, WebArena's repository has directed users toward subsequent infrastructure improvements through AgentLab and BrowserGym [6]. Research discussions should account for those changes rather than criticizing the earliest implementation indefinitely.

A public test can also become an optimization target. Researchers may repeatedly tune prompts, tools, and model choices against known tasks. Even without direct training contamination, development choices can overfit to a benchmark's quirks. When examples circulate publicly, the distinction between memorized patterns and transferable skill becomes harder to establish. That does not make public benchmarks useless. It means public scores should be paired with evaluation on independent task distributions and transparent reporting of test exposure.

The practical solution is a layered evidence system. Keep public benchmarks for comparability. Use training environments for skill acquisition and controlled ablations. Reserve independent, held-out evaluations for generalization claims. For high-stakes workflows, add safety and durability checks that ordinary task-completion metrics may miss.

4. The anatomy of useful RL environments for AI agents

A useful environment begins with a capability hypothesis. 'Improve coding' is too broad to guide task design. 'Recover a broken dependency installation without altering unrelated packages' is more testable. The designer can specify the initial state, permitted tools, hidden causes, expected behavior, prohibited shortcuts, and observable outcomes. A collection should vary the underlying causes rather than merely rewriting the task description.

Task construction must separate an objective from its reference solution. An oracle solution can demonstrate feasibility and help validate the grader, but agents should not be rewarded for imitating one trajectory when several valid solutions exist. Outcome-based checks should accept equivalent repairs and reject brittle hacks. For example, a service may pass a superficial health check while losing persistence across restart. A durable verifier should test both the immediate state and relevant subsequent behavior.

Environment packaging matters. Dependencies, operating-system versions, resource limits, fixture data, and network permissions affect outcomes. Containerization can support reproducibility, although containers do not eliminate nondeterminism from timing, external services, or model sampling. A robust package records versions and tests that a clean reset reproduces the expected initial conditions.

Safety constraints should be explicit. An agent must not receive a high score for accomplishing a task by deleting unrelated records, bypassing authentication, or modifying the evaluator. This is particularly important when rewards are automatically computed. A verifier needs negative controls: known-wrong solutions that should fail. Adversarial review should look for unintended paths to a passing score.

Finally, useful environments capture evidence. A binary pass/fail label is helpful for ranking, but a trajectory can show whether the agent diagnosed the problem, tried a risky command, recovered from an error, or merely got lucky. Trajectory analysis can inform curriculum design and debugging without requiring every intermediate action to become a reward. The distinction between evaluation evidence and training reward should remain clear.

5. Diversity, curriculum, and the problem of transfer

A large task count is not the same as a broad task distribution. Hundreds of tasks may share a template, dependency, or hidden exploit. A model trained on them may learn the shortcut rather than the intended capability. Diversity should therefore be described in terms of mechanisms: types of failures, state representations, tool affordances, ambiguity, time horizons, dependency structures, and required verification.

Curriculum design introduces tasks in an order that supports learning. Early tasks might isolate one tool operation; later tasks might require diagnosis, planning, execution, and validation across several subsystems. Difficulty can be measured empirically through repeated model-agent trials, but those measurements depend on the model and scaffold used. A task that is difficult for one agent may be trivial for another. Difficulty labels should be versioned rather than treated as universal truths.

Benchmark generalization is ultimately a question of transfer. Transfer is the central test. Suppose a model learns to repair misconfigured service ports in a training suite. An independent evaluation might vary the service framework, file layout, error message, and deployment workflow while preserving the underlying diagnosis. Success would offer stronger evidence of generalized reasoning than improvement on near-duplicate tasks. A harder evaluation might test an unseen failure mechanism entirely; that probes adaptability, not merely familiar pattern application.

Designers should also distinguish surface variation from causal variation. Renaming variables or changing a color scheme may prevent exact memorization while leaving the reasoning identical. Conversely, changing a dependency graph or introducing a conflicting constraint can alter the problem's causal structure. Both kinds of variation are useful, but they answer different questions.

No environment collection can perfectly reproduce deployment. Production systems contain organizational constraints, changing data, permissions, social context, and unpredictable failures. The goal is not total simulation. It is to identify which features of the deployment task matter for learning and measurement, then test whether performance survives when irrelevant features change and relevant constraints remain.

6. A practical AI model evaluation architecture

An effective AI agent evaluation program can be organized into three non-overlapping collections. The first is a training set of executable environments with reward signals suitable for the chosen optimization method. The second is a development set for debugging prompts, tools, and agent behavior. The third is a held-out evaluation set that is protected from training and repeated tuning. The distinction is conceptual; organizations should document their exact access policies and versioning.

For each collection, record task provenance, creation date, software versions, domain, capability tags, difficulty evidence, known failure modes, and evaluation code. This metadata helps explain performance changes and detect accidental overlap. Similarity audits should look beyond identical strings to shared source issues, fixtures, solutions, and causal templates.

Measure more than aggregate success. Useful dimensions include task completion, constraint violations, time or compute consumed, repeated-run consistency, recovery after errors, and durability after restart. Not every dimension belongs in a single composite score. A weighted score can conceal serious safety regressions if improvements elsewhere compensate numerically. Report critical constraints separately.

Repeated trials are important because agent outcomes can vary with model sampling and tool timing. A single run may not represent expected behavior. Teams should report sample counts and uncertainty rather than presenting small differences as definitive. Comparisons also require matched conditions: identical budgets, environment versions, available tools, and scoring criteria where possible.

A research team should investigate failures before scaling a dataset. If most failures result from ambiguous instructions or broken setup scripts, adding more tasks may multiply noise. If agents consistently fail at state tracking, targeted tasks that require revisiting earlier observations may be valuable. The environment program should respond to diagnosed capability gaps, not just the demand for more examples.

7. Illustrative experiment: does environment diversity improve reliability?

The following is a proposed experiment, not a completed Deaimer study. Consider two agent-training collections with matched task counts and approximate compute budgets. Collection A contains many variations of a narrow repository-repair template. Collection B contains fewer repeated templates and a wider range of causal failure mechanisms, including dependency conflicts, state corruption, permissions, and service lifecycle problems.

The hypothesis is that Collection B will produce better transfer on held-out tasks whose surface details and failure mechanisms differ from training. The independent variables are task distribution and diversity; the dependent measures include held-out success, constraint violations, and consistency across repeated runs. The comparison should control for model initialization, agent scaffold, optimization procedure, reward budget, and evaluation environment.

A strong design would include at least three evaluation slices. A near-distribution slice tests familiar mechanisms with new surface details. A mechanism-transfer slice introduces different combinations of familiar components. An out-of-distribution slice tests unseen mechanisms. Results should be disaggregated because improvement on one slice does not establish universal superiority.

Before running the experiment, researchers should predefine exclusion rules for invalid tasks, how many agent trials will be performed, and how uncertainty will be estimated. They should also audit whether one collection accidentally includes easier reward signals or more informative error messages. Otherwise a difference attributed to diversity might actually be caused by feedback quality.

Possible outcomes include no difference, better near-distribution performance for the narrow collection, or improved held-out transfer for the diverse collection. All would be informative. The value of the experiment is that it turns a plausible design principle into a falsifiable claim rather than a marketing assertion.

8. What model developers and enterprise buyers should ask about agent reliability

Model developers should begin with a capability map, not a dataset catalog. Which real workflows are failing? Is the bottleneck perception, planning, tool use, state tracking, domain knowledge, or recovery? Which failures are expensive or unsafe? Answers determine what environments should contain and what evaluators must check.

Enterprise buyers face an additional question: whether an environment is representative of their operating constraints. A task may be technically realistic but irrelevant to the buyer's software stack or permission model. A controlled pilot can reveal whether tasks execute reliably, distinguish good from bad behavior, and cover the intended failure modes. Procurement decisions should consider ownership and licensing terms, provenance, security boundaries, version maintenance, and the ability to independently inspect grading behavior.

Evidence of quality is more valuable than a promise of quantity. Request a small sample with documented initial states, objective checks, negative controls, and evidence of repeated execution. Examine whether multiple valid solutions pass, whether destructive shortcuts fail, and whether results are reproducible. Then assess whether the task mix matches the target capability. A thousand nearly identical tasks may be less useful than a smaller, carefully varied collection, although the relative value must be tested for the actual training setup.

Organizations should resist treating a benchmark rank as a procurement guarantee. The appropriate question is not simply 'Which model scores highest?' but 'Which model-agent configuration performs acceptably on our tasks, under

our constraints, at our operating cost?' Public results can narrow candidates; controlled private evaluation can support a deployment decision.

9. Where Deaimer fits

Deaimer publicly describes work on custom RL environments, coding-agent evaluations, and independent benchmarks, including resettable task environments, automated graders, reference solutions, adversarial checks, and model-run analysis [7]. Its ColdStart benchmark focuses on autonomous recovery of broken systems while checking safety, durability, and integrity alongside functional recovery [8]. These descriptions provide a relevant product context; they do not establish that any particular Deaimer dataset improves model performance without a controlled experiment.

For teams developing coding agents or operational agents, the useful connection is the ability to commission or evaluate tasks around a specific capability rather than rely only on public benchmark coverage. Deaimer's published RL-environments page describes a controlled pilot for constructing or independently auditing a small batch under agreed requirements [7]. That is a reasonable first step when task relevance and verifier integrity matter more than a headline task count.

The broader research principle remains independent of any vendor: training environments, public benchmarks, and held-out evaluation have different jobs. Treating them as complementary instruments makes the evidence about agent reliability more credible.

Conclusion: build evidence, not just scores

The next stage of agent research requires a richer notion of progress. Public benchmarks remain essential because they establish shared tasks and expose failure modes. Realistic RL environments are valuable because they can provide repeated, consequential interactions in which agents learn to act. Independent evaluations matter because they test whether the resulting behavior transfers beyond the training distribution.

None of these components is sufficient alone. A realistic environment with a weak verifier can teach shortcuts. A carefully curated benchmark can become stale or overexposed. A high aggregate score can conceal failures on rare but important constraints. Reliability emerges from the design of the full development loop: representative tasks, valid rewards, reproducible execution, controlled exposure, repeated measurement, and explicit attention to safety.

For researchers, the next useful experiment is to test which environment properties actually improve transfer. For model developers, it is to identify the causal skills their agents lack. For buyers, it is to demand evidence that an environment collection measures and trains the behavior they need. The question is not how high an agent can climb on a leaderboard. It is what that score allows us to predict about the work the agent will do next.

References

- [1] Liu, X., et al. AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688 (2023; revised 2025).
<https://arxiv.org/abs/2308.03688>
- [2] Zhou, S., et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. ICLR 2024.
<https://www.cmu.edu/flame/research/2024/webarena.html>
- [3] Xie, T., et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. NeurIPS 2024. <https://arxiv.org/abs/2404.07972>
- [4] OpenAI. Introducing SWE-bench Verified. August 13, 2024; updated February 24, 2025.
<https://openai.com/index/introducing-swe-bench-verified/>
- [5] OpenAI. Why SWE-bench Verified no longer measures frontier coding capabilities. February 23, 2026.
<https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/>

- [6] WebArena official repository. December 2024 update on AgentLab and BrowserGym. <https://github.com/lhyscau/webarena>
- [7] Deaimer. RL Environments & Coding-Agent Evaluations. Accessed October 12, 2026.
<https://www.deaimer.com/rl-environments>
- [8] Deaimer. ColdStart — Audited AI Agent Recovery Benchmark. Accessed October 12, 2026.
<https://www.deaimer.com/benchmarks/coldstart>