arXiv:2607.27283v1 [cs.LG] 29 Jul 2026

Benchmarking the Residual: What Long-Horizon Evaluations Add Beyond Matched Short-Task Performance

Chao Peng[*] [†] **Zhiheng Lyu**[*] **Peijie Dong**[*] **Hande Dong**[†] **Qiang Lin**
Tencent
chao.peng@acm.org, zl149@illinois.edu, peytondong@tencent.com
handedong@tencent.com, cheaterlin@tencent.com

Abstract

Long-horizon benchmarks often show that agents fail more as tasks become longer. This observation is useful for deployment, but it does not by itself explain why failure occurs. More stages create more opportunities for ordinary errors to compound; longer tasks may also contain harder individual decisions or become harder as conversation history, tool outputs, and environment changes accumulate. We use *trajectory-induced degradation* to mean this last possibility: earlier execution makes later work harder. When the harmful accumulation is specifically the text visible to the model, it is often called *context rot*. In this position paper, we argue that to claim a "long-horizon failure", benchmarks must compare actual full-task success against a baseline prediction built from short, individual stages. We call the log-ratio between this prediction and actual success the *horizon residual*. The comparison must use the same agent configuration and specify in advance how stages, checkpoints, information, and budgets will be chosen. The residual shows that the full rollout differs from the chosen baseline; targeted experiments are still needed to explain why.

The key idea. Benchmark design: Longer can mean more work, not harder work. Benchmarks should distinguish task size from how hard each stage is and how strongly stages depend on one another. **Result interpretation:** More stages create more chances to fail. Compare actual full-task success with a prediction built from matched short stages; the *horizon residual* summarizes the mismatch, not its cause.

1 Introduction

Language-model agents are increasingly evaluated on tasks that extend beyond a single response or isolated code edit. In software engineering, recent benchmarks range from repository-level issue resolution [Jimenez et al., 2024] to release-sized evolution [Le et al., 2025], chained package upgrades [Lam et al., 2026], and continuous milestone execution on a persistent codebase [Deng et al., 2026]. Related interactive benchmarks require agents to navigate websites, operate desktop applications, coordinate APIs, or conduct iterative research engineering [Zhou et al., 2023, Xie et al., 2024, Trivedi et al., 2024, Wijk et al., 2024]. Across these settings, success generally becomes less reliable as the amount of work, the depth of inter-stage dependence, and the accumulated execution history increase. This trend matters for deployment because many valuable tasks require sustained planning, tool use, state tracking, verification, and recovery over extended trajectories.

The phrase *long horizon*, however, can refer to several different quantities:

[*] These authors contributed equally.
[†] Corresponding authors: chao.peng@acm.org and handedong@tencent.com.

Preprint.

- the minimum number of effective actions required;
- the number of separately verifiable milestones;
- the depth of the dependency chain;
- the duration of the agent's rollout; or
- the amount of prior history visible to the model.

These quantities often move together in naturally collected tasks, but they need not. A task may contain many independent operations with shallow dependencies, or only a few stages that strongly depend on earlier state. A short task may also produce a long trace because the agent retries or explores irrelevant actions. Treating these cases as one notion of length obscures what a benchmark actually measures [Wang et al., 2026, Kwa et al., 2025].

A lower end-to-end success rate on a longer task is therefore not yet a diagnosis of a long-horizon failure mechanism. Even if each local stage is unchanged, requiring more stages to succeed can reduce task success through ordinary error compounding. Longer tasks may also contain intrinsically harder local decisions, expose the agent to noisier context, or allow earlier actions to alter the state on which later decisions depend. Existing results illustrate these possibilities: independent and continuous milestone protocols can produce sharply different outcomes [Deng et al., 2026]; agents can degrade when they inherit their own prior code changes [Jin et al., 2026]; and controlled history manipulations show sensitivity to erroneous prior outputs at a matched evaluation turn [Sinha et al., 2026]. Because these factors often vary together, a raw performance curve can establish a deployment limitation without identifying its source.

This distinction separates two legitimate purposes of evaluation. A deployment benchmark asks whether the complete system succeeds under realistic budgets and interaction rules. A diagnostic benchmark asks which controlled contrast explains behavior that a simpler model of local competence does not. The first purpose prioritizes ecological validity; the second requires replayable states, explicit interventions, and assumptions that can be audited. Neither replaces the other. In practice, however, long-horizon results are often given diagnostic readings that the deployment observation alone cannot license.

We therefore propose a simple diagnostic comparison. First, run the same agent on short, verifiable stages from declared checkpoints. Next, combine those stage results into a prediction for the full task. Finally, compare the prediction with success in a natural end-to-end rollout. We use *counterfactual* for this pre-specified alternative evaluation protocol, not for an individual-level causal claim.[1] The horizon residual summarizes how far the natural rollout departs from the checkpoint-based prediction. It is not a causal answer, but it can direct follow-up experiments on history, state, planning, verification, or recovery.

This position paper makes three contributions:

1. We **advocate** treating raw performance decline as a descriptive deployment result and the counterfactual residual as a separate, protocol-dependent diagnostic.
2. We **propose** a deliberately simple and auditable product baseline, together with conditional-model and checkpoint-simulation extensions when its assumptions fail.
3. We **articulate** reporting requirements, scope conditions, and an intervention agenda for turning residual patterns into testable causal hypotheses.

2 Diagnosing Performance Decline Beyond "Longer Is Harder"

Consider a coding agent tasked with making four independent repairs. Suppose it succeeds on each repair with probability 80% when evaluated separately. If these outcomes compose independently, then even when no additional long-horizon failure occurs, the probability of completing all four repairs is only $0.8^4 \approx 41\%$. Thus, a low end-to-end success rate is not by itself evidence of a distinctive long-horizon limitation; it may follow from ordinary error compounding.

[1] The resulting quantities compare evaluation protocols; they are not individual-level potential-outcomes counterfactuals.

Now suppose the same agent completes the four-repair task only 10% of the time. The gap between 41% and 10% is more informative. Earlier edits may contaminate the repository state, long tool transcripts may distract the model, or the agent may lose track of the original goal, omit intermediate verification, or fail to recover from an early mistake. These are plausible explanations, but the raw 10% success rate cannot resolve which, if any, is responsible. Interpretation therefore requires a declared model of what matched short-stage performance predicts.

Table 1 summarizes three effects that are easy to conflate:

Table 1: Distinct sources of performance degradation as task length increases, together with the control required to distinguish each source.

Effect	**What changes?**	**Useful control**
Error compounding	The local stages are no harder, but more of them must succeed.	Measure each matched stage and compose the estimated success probabilities.
Harder local work	Longer tasks contain more ambiguous, specialized, or poorly verified stages.	Match the distribution of local stage difficulty.
History dependence	Later steps become harder as context, state, or earlier errors accumulate.	Compare retained history with reset, compressed, or repaired history.

Hereafter, *trajectory-induced degradation* means that accumulated execution (the transcript, tool outputs, environment state, or earlier errors) makes later work harder. *Context rot* is the narrower case in which the growing visible text is the source of the decline [Hong et al., 2025]; positional degradation in long contexts is a related observation [Liu et al., 2024]. These labels describe where performance changes, not the underlying mechanism. They may cover distraction from a noisy transcript, cross-module planning, state contamination, or failed recovery.

Recent evidence shows why these categories should not be collapsed. In a controlled running-state task, Sinha et al. [2026] hold the evaluation turn and history format fixed while varying the error rate of the preceding outputs. Cleaner histories improve subsequent accuracy, showing that the content of the execution history matters beyond its length. A related content-versus-length control appears in repeated social-dilemma games: holding prompt length fixed while replacing the visible history with synthetic cooperative records substantially restores cooperative behavior, isolating history content rather than length as the trigger [Liu et al., 2026]. ChainSWE provides a complementary repository-level comparison: across 100 chains containing 304 issues from 54 projects, average per-bug accuracy falls from 58.9% when each bug begins from an oracle-correct repository state to 36.5% when the agent inherits its own prior modifications [Jin et al., 2026]. Moreover, 318 of 663 downstream sequential failures in its baseline setting occur on bugs that the same systems solve from the oracle state. These results do not establish a universal horizon law, but they illustrate the diagnostic value of comparing natural execution against a matched state-repaired counterfactual.

3 A System-Matched Counterfactual for Long-Horizon Evaluation

3.1 Decomposition and Checkpoints

We propose beginning with a pre-specified decomposition of the long task into verifiable stages. In a software engineering task, these stages might be "find the bug", "change the data model", "update the API", and "pass the integration tests". Each stage should be semantic: it should be defined by an acceptance condition rather than by an arbitrary number of turns or tool calls.

The next step is to run the same deployed agent from declared checkpoints (Figure 1). Here, the deployed agent denotes the complete system, including the model, prompt, tools, memory, and execution framework. Let S_i denote success at stage i, and let I denote a pre-specified checkpoint protocol: the environment state, visible history, revealed information, and local budget supplied at that stage. Along a fixed path on which every stage is required, define $q_i = \Pr(S_i = 1 \mid S_{<i} = 1, I)$. Here, q_i is the agent's chance of passing stage i when the required earlier stages are already complete and the benchmark starts the agent from its declared checkpoint.

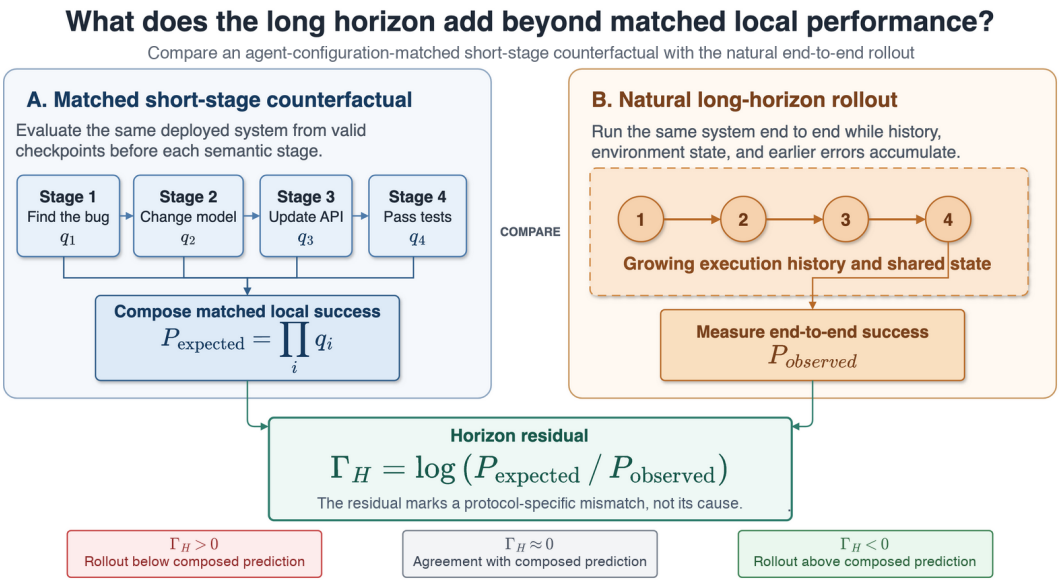

Figure 1: Agent-configuration-matched comparison between checkpoint-based local evaluation and natural end-to-end execution. Declared checkpoint interventions estimate the conditional stage probabilities used to construct P_{expected}, whereas the natural rollout yields P_{observed} under accumulated history and shared state. The horizon residual Γ_{H} records their protocol-specific log contrast; mechanism claims require additional interventions.

3.2 The Product Baseline

The chain rule then gives the auditable baseline

$$P_{\text{expected}} = \prod_i q_i. \tag{1}$$

In simple terms, if an agent has a 90% chance of passing Stage 1 and an 80% chance of passing Stage 2 from clean, compatible states, we predict a 72% overall success rate (0.90×0.80). If the actual agent succeeds only 30% of the time, the intuitive shortfall is 42 percentage points; the horizon residual defined below reports this same comparison as a log-ratio.

The homogeneous special case clarifies why this baseline matters. If every stage succeeds with a constant probability p, failures are independent, and no recovery is possible, then an n-stage task succeeds with probability p^n. Equivalently, the first horizon at which success falls below a threshold s is approximately $H_s = \lceil \log(s)/\log(p) \rceil$ [Sinha et al., 2026]. This analysis shows that a substantial raw decline can arise without any additional long-horizon mechanism. Its assumptions are deliberately strong: stage accuracy may change with position, errors may alter future state, and natural rollouts may recover. We therefore use p^n as an auditable null model rather than a description of how agents necessarily behave.

Three baseline forms cover different task structures. The *product baseline* multiplies conditional stage-success rates when the checkpoints form one compatible path. An *oracle-scaffolded baseline* multiplies success rates from separately chosen, clean canonical states; because those states may not arise in one real rollout, it requires extra compatibility and independence assumptions. A *checkpoint simulator* is needed when solutions branch, failures can be retried or recovered from, or one stage changes the state and odds of later stages.

For a heterogeneous task family, predictions should be composed within each task and then aggregated using the same task weights as P_{observed}; multiplying stage averages can give a different quantity. In every case, the benchmark should declare the baseline model, estimate it from the corresponding checkpoint behavior, and compute its prediction before evaluating the natural end-to-end rollout.

Checkpoint evaluation also raises a cross-world composition problem. An oracle reset, repair of an agent-produced state, fresh context, and revelation of a local goal are separate interventions and therefore define different estimands.

Moreover, an alternative valid solution at stage i need not be composable with a canonical downstream checkpoint constructed from a reference solution. A benchmark should declare whether it evaluates a canonical path, constructs solution-conditioned downstream states, or simulates transitions across compatible states; otherwise the q_i values may describe stages that cannot be combined in one coherent rollout.

Existing results suggest that checkpoint and continuous protocols can differ sharply. SWE-Milestone evaluates 98 human-verified milestones across seven repositories and five programming languages. The reported scores are approximately 80% or higher when milestones are attempted independently from canonical snapshots, whereas the best reported continuous-evolution score is 38.03% [Deng et al., 2026]. On scikit-learn, one reported agent–model configuration drops from 93.2% independently to 21.1% continuously. This is not yet the compositional prediction in Equation 1: independent milestone scores must still be composed along the relevant task graph, with compatibility assumptions made explicit. It nevertheless shows that reset-state and persistent-rollout protocols estimate materially different quantities.

3.3 The Horizon Residual

Let P_{observed} denote success under the natural end-to-end rollout. For a fixed declared counterfactual protocol, we summarize the comparison as

$$\Gamma_{\text{H}} = \log \frac{P_{\text{expected}}}{P_{\text{observed}}}, \tag{2}$$

where $\log$ denotes the natural logarithm, so Γ_{H} is measured in nats. We refer to Γ_{H} as the *horizon residual*: a protocol-specific log contrast, not a direct measure of a horizon mechanism. In finite samples, it should be reported with raw counts, uncertainty, and an explicit treatment of zero or near-zero estimated probabilities. Relative to the specified counterfactual:

- $\Gamma_{\text{H}} > 0$ indicates lower success in the natural rollout than the model predicts;
- $\Gamma_{\text{H}} \approx 0$ indicates agreement at the resolution supported by the data;
- $\Gamma_{\text{H}} < 0$ indicates higher success in the natural rollout than the model predicts.

Section 3.4 gives the joint reading of these cases with both rates' magnitudes.

When the natural rollout can be evaluated at the same semantic boundaries, the contrast admits a useful stage-wise decomposition. Let $r_i = \Pr(S_i = 1 \mid S_{<i} = 1, N)$ denote conditional stage success under the natural-rollout protocol N. If success requires every stage on a fixed path, then

$$P_{\text{observed}} = \prod_i r_i, \qquad \Gamma_{\text{H}} = \sum_i \log \frac{q_i}{r_i}. \tag{3}$$

This breakdown shows which step in the natural rollout had the largest drop relative to the checkpoint baseline. A positive term marks a stage at which success is lower in the natural rollout. It does not identify why the difference occurs. Moreover, a positive term at stage i marks where the mismatch *surfaces*, not necessarily where it *originates*: degradation accumulated earlier may become visible only at stage i. Estimating later r_i can also be expensive because only rollouts with successful prefixes contribute to the corresponding probability. These nested samples are correlated across stages, so intervals for the stage-wise terms should be obtained by bootstrap over complete task units rather than from the analytic approximation below (Equation 4). Branching paths and recoverable failures again require a transition model rather than this fixed-path factorization.

The four-repair example of Section 2 makes the computation concrete. Matched checkpoints that give $q_i = 0.8$ on each repair yield $P_{\text{expected}} = 0.8^4 \approx 0.41$; a natural rollout that succeeds 10% of the time then gives $\Gamma_{\text{H}} = \log(0.41/0.10) \approx 1.41$ nats. The natural rollout underperforms the composed local prediction by a factor of about four. Stage-wise terms of, say, $(0.2, 0.5, 0.6, 0.1)$ would indicate

that the mismatch surfaces mainly at the third repair and would target the first intervention there, subject to the surfacing-versus-origin caveat above.

A first-order uncertainty calculation further clarifies the data requirement. Suppose $\widehat{q}_i$ is estimated from m_i independent Bernoulli trials and $\widehat{P_{\text{observed}}}$ from m_0 independent end-to-end trials. Ignoring covariance, the delta method gives

$$\operatorname{Var}(\widehat{\Gamma_{\text{H}}}) \approx \sum_i \frac{1-q_i}{m_i q_i} + \frac{1-P_{\text{observed}}}{m_0 P_{\text{observed}}}. \tag{4}$$

Thus, uncertainty grows rapidly when any local or end-to-end success probability approaches zero. In realistic evaluations, estimates may share tasks, seeds, checkpoints, or agent states, violating the independence approximation in Equation 4. Such sharing typically induces positive covariance between checkpoint and rollout estimates on the same task, so the independence approximation understates the variance of $\widehat{\Gamma_{\text{H}}}$. We therefore recommend paired or hierarchical bootstrap intervals over complete task units, with the analytic expression used only as intuition and a planning approximation.

The residual is diagnostic, but it is not by itself a causal explanation. A positive value marks a mismatch with the selected local-performance model, not the reason for that mismatch. If history-induced degradation is the hypothesis, the benchmark can rerun the same task with periodic context reset or compression while holding other protocol features fixed. Each intervention should define its own $P_{\text{expected}}(I)$ and $P_{\text{observed}}(I)$ before residuals are compared; keeping the original counterfactual fixed instead answers a different deployment question. A selective change in the contrast is more focused evidence than another natural-rollout curve, although remaining intervention confounds still require analysis.

3.4 Interpreting the Residual

The expected and observed success rates answer distinct questions and should be interpreted jointly, always relative to the specified counterfactual:

- **Both are high:** both protocols place the task within the system's measured capability under the stated budgets and information.
- **Both are low:** the task may remain a useful deployment stress test, but the comparison has little power to separate counterfactual-model error, compounding, and history-dependent effects.
- **Expected success is high but observed success is low:** the natural rollout has an unexplained deficit under the selected model, motivating targeted interventions and sensitivity checks rather than an immediate mechanism claim.
- **Observed success exceeds expected success:** the natural rollout may exploit feedback, redundancy, adaptation, or recovery omitted by the counterfactual; interventions are still needed to determine which explanation, if any, accounts for the advantage.

If Equation 1 predicts near-zero success, the log contrast is statistically unstable and an all-failure outcome is diagnostically weak. Informative benchmark regimes should keep the counterfactual prediction away from degenerate extremes where feasible, report partial progress and cost, and include interventions that test why the two protocols differ.

3.5 Choosing the Baseline

The appropriate baseline depends on the scientific question. Fresh context tests a different estimand from history-matched checkpoints, which may be more relevant to deployment forecasting because they retain the expected burden of long context. State repair, as in ChainSWE's oracle mode, keeps the position in the task sequence while replacing earlier agent modifications with verified ones [Jin et al., 2026]; oracle reset may additionally replace other aspects of state. Revealing a stage goal changes the information available to the agent and should not be treated as a pure state reset. No single baseline serves every estimand. Benchmarks should therefore declare checkpoint state, state-repair policy, visible history, revealed information, and local budget because each choice changes the meaning of P_{expected} and Γ_{H}.

4 Separating Task Structure from Local Difficulty

The key contrast is concrete: one hundred independent renames involve much work but little dependency, while a schema–storage–API–client migration may have fewer stages but require each stage to build on the last. The number of turns does not capture this difference: one shell command may modify hundreds of files, whereas a difficult localization problem may consume many turns without adding semantic work.

We therefore advocate reporting two complementary annotations:

- **Work exposure N_{work}:** how many required, separately verifiable stages the benchmark declares. More stages create more opportunities for local failure.
- **Dependency depth H_{dep}:** how many stages lie on the deepest required chain. Greater depth means that later work must build on more earlier state or feedback.

Neither N_{work} nor H_{dep} is an intrinsic property of a task: both depend on the chosen decomposition, acceptance conditions, and dependency graph. They can nevertheless vary independently within a declared design. Repetitive work can also be structurally shallow while becoming dependent during execution through a shared context window.

HORIZON makes a related distinction by defining an agent-independent intrinsic horizon H^* as the minimum number of effective actions required by an optimal policy and by constructing longer task families through depth and breadth extensions [Wang et al., 2026]. This is a useful safeguard against calling a short task "long-horizon" merely because an agent repeats failed actions. At the same time, H^* requires an oracle, expert demonstration, or formal task specification, and its operational mapping differs across domains. We therefore view intrinsic horizon, annotated work exposure, and dependency depth as complementary descriptions: the first estimates minimal action length, while the latter two expose how a declared evaluation decomposes and connects verifiable work.

Local difficulty should likewise be measured rather than declared. A task is not simply "hard"; it is hard *for a particular agent operating with particular information, tools, verifiers, and budgets*. The same refactor may be easy when a failing test localizes the problem and difficult when feedback is delayed until final integration. For this reason, measured local success is more informative than a human-assigned easy/hard label.

Human completion time is useful for deployment calibration but does not resolve this confound. Kwa et al. [2025], for example, define the task-completion time horizon at a 50% success rate and estimate a horizon of roughly 50 minutes for a frontier system on their task mixture. The metric communicates the scale of work that a system can reliably complete, but two tasks with the same human duration may differ substantially in annotated work exposure, dependency depth, verifier quality, and agent-local difficulty. Time horizon and the protocol-specific structural annotations proposed here therefore serve complementary purposes. Table 2 sketches the factorial conditions that these two annotations induce.

Table 2: A factorial coding-task design that separates dependency depth from measured local-stage difficulty. Work exposure and local success should be matched within each intended comparison.

	Easier local stages	**Harder local stages**
Shallow chain	Independent repairs with diagnostic tests	Independent repairs with ambiguous failures or security constraints
Deep chain	An explicit migration with strong intermediate checks	A multi-module migration with sparse feedback and difficult recovery

Task length and local difficulty will often be correlated in naturally collected tasks. They need not be statistically independent, but a benchmark can attempt to separate them experimentally by varying the declared dependency structure while approximately matching local success, and by varying local difficulty while holding that structure fixed.

5 A Benchmark Design Protocol for Coding and Terminal Agents

Coding and terminal environments are particularly well suited to this design. Repository snapshots and containers make earlier states replayable, while tests and state invariants provide semantic acceptance conditions. We propose the following compact workflow:

1. **Pre-register.** Declare the task, stages, acceptance conditions, dependency graph, rollout policy, tools, verifier, and recovery rules before evaluation.
2. **Define checkpoints.** Record the environment state, visible history, revealed information, repair policy, and rules for alternative valid implementations.
3. **Match budgets.** Report token, tool, time, verification, and retry costs for both local and end-to-end runs. Where possible, include both equal per-stage budgets and an aggregate-budget-matched comparison; otherwise count the resource difference as part of the intervention.
4. **Audit local performance.** Report per-stage successes, trial counts, uncertainty, and heterogeneity; pre-specify how zero or near-zero estimates will be handled.
5. **Choose the model.** Use the product baseline for a fixed compatible path. Use a conditional model or checkpoint simulator when the task includes branches, retries, recovery, or shared-state dependence.
6. **Run and intervene.** Estimate P_{expected} before measuring P_{observed}, then test targeted changes such as history reset, state repair, stronger verification, rollback, or explicit planning.
7. **Test sensitivity.** Recompute predictions and residuals under multiple reasonable decompositions, dependency graphs, and admissible checkpoint protocols. Call a conclusion robust only when its sign is consistent across the pre-declared set; otherwise report the range of Γ_{H}.
8. **Audit the benchmark.** Report task-inclusion criteria, exclusions, leakage checks, and contamination risks for both the agent and any decomposition or grading procedure.

An informative task family should contain both shallow and deep variants. Independent edits provide a high-exposure, shallow control. A causally ordered patch stack (for example, schema, then storage, then API, then client) provides a deep condition. The local stages should be selected so that their measured success distributions are comparable. Both stage-given and goal-only variants are useful because a checkpoint may reveal localization or decomposition information that is unavailable in the natural task.

History-induced degradation also warrants a dedicated controlled comparison, matching the useful control of Table 1. The agent can perform the same sequence of simple operations while retaining the full history, periodically compressing it, or restarting from a clean context. Such a comparison helps estimate the total effect of the declared history intervention; it does not identify a single underlying mechanism. Observational evidence underscores why the controlled version matters: in an ultra-long-horizon software benchmark, none of the 71 rollouts that used mid-trajectory context compaction passed, against 8.9% without it [Desai et al., 2026]; absent a matched intervention, such a correlation cannot distinguish compaction as a cause of failure from compaction as a marker of rollouts that were already failing. Environment state, revealed information, and budgets must be held fixed across conditions unless their joint intervention is the stated estimand.

6 Related Evidence

The literature suggests a hierarchy of evidential strength. End-to-end curves answer what happened under a deployment protocol; trajectory labels and dense rewards add localization [Lightman et al., 2024]; matched checkpoints test whether the natural rollout exceeds local prediction; and targeted interventions test whether a proposed factor changes that contrast [Sinha et al., 2026, Wang et al., 2026]. Figure 2 summarizes this progression. The levels are cumulative: mechanism experiments do not remove the need to report raw success, cost, and partial progress.

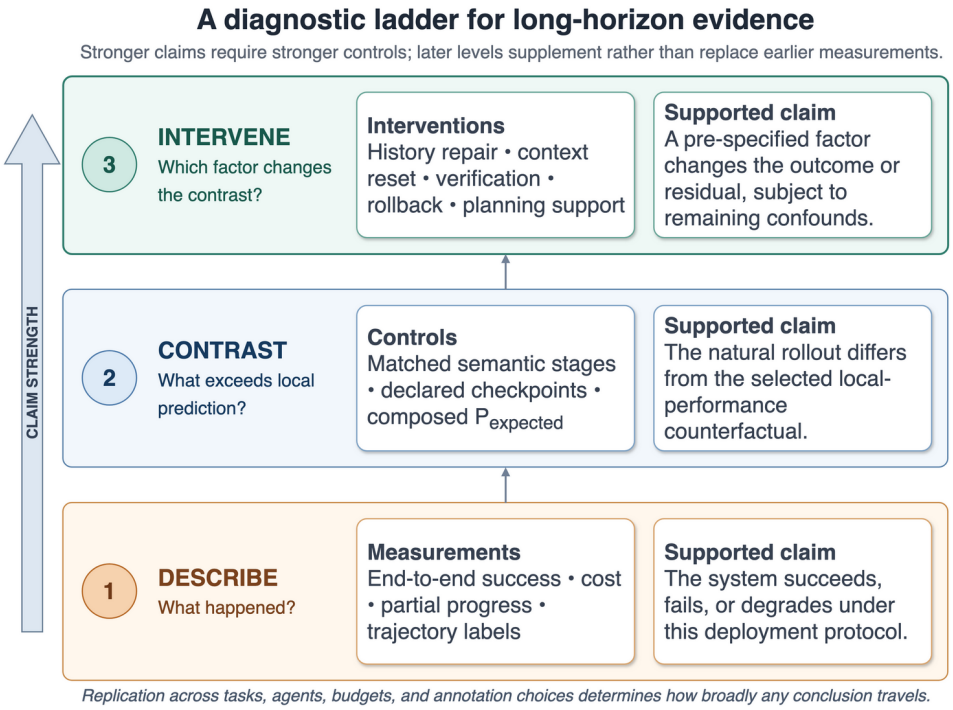

Figure 2: Hierarchy of evidence for long-horizon evaluation. Descriptive measurements characterize deployment behavior; a declared local-stage counterfactual identifies mismatch with matched competence; and targeted interventions test scoped mechanism hypotheses. Replication across tasks, systems, budgets, and annotation choices determines the generality of each conclusion.

Table 3 highlights how prior benchmarks observe performance drops on long tasks but do not yet combine all the history and state controls needed to isolate why the drop occurred. The studies differ in whether they control local solvability, repair prior state, or intervene on history.

Table 4 restates this comparison as a component-level audit of the proposed sequence. Every component exists in some published evaluation, and we claim no novelty for any of them individually. The position is that no reviewed study implements the full sequence (matched local measurement, declared composition, natural rollout, state-compatibility handling, targeted intervention, and protocol-sensitivity analysis) under one pre-specified protocol. Partial marks record, for example, ChainSWE's single oracle-repair intervention, SWE-Milestone's inferred rather than declared dependency graph, and the homogeneous p^n analysis of Sinha et al. [2026] on a controlled synthetic task rather than a natural deployment rollout. If a prior evaluation is found to implement the complete sequence, the claim of this paper narrows to advocacy of its standardization.

From isolated issue resolution to sustained software evolution. SWE-bench established repository-level issue resolution as a practical testbed for language models [Jimenez et al., 2024]; SWE-Gym and OpenHands contributed executable training tasks, trajectories, and a general agent platform [Pan et al., 2025, Wang et al., 2025]. SWE-Lancer and time-calibrated evaluations broaden ecological validity through freelance work and human-duration estimates [Miserendino et al., 2025, Kwa et al., 2025]. SWE-Bench Pro, SWE-EVO, and NL2Repo-Bench extend scope toward enterprise issues, release-sized changes, and full-repository construction [Deng et al., 2025, Le et al., 2025, Ding et al., 2025]. Their results indicate that current systems struggle as scope and coordination demands grow. Their reported protocols are not designed to hold local difficulty, planning burden, feedback sparsity, and work exposure fixed while varying horizon, so they motivate rather than instantiate the counterfactual proposed here.

Table 3: Representative empirical evidence, the history or state control implemented by each study, and the diagnostic question that remains unresolved. Reported values follow study-specific protocols and are not directly comparable across rows.

Study	Reported evidence	History/state control	What remains unresolved?
SWE-EVO [Le et al., 2025]	48 release-sized tasks modify 20.9 files on average and are checked by 874 tests on average.	Each task starts from its prescribed release state.	Its reported protocol is not designed to separate scale, local difficulty, and horizon.
NL2Repo-Bench [Ding et al., 2025]	Across 104 repository-generation tasks, the best reported mean test pass rate is 40.2%; no more than five repositories are fully completed.	One persistent workspace per task.	The reported protocol does not include a matched-stage prediction to distinguish error compounding from failures of cross-stage coherence.
SWE-Chain [Lam et al., 2026]	12 release chains contain 155 transitions and 1,660 grounded requirements.	Code persists across transitions; conversations reset.	The reported transition estimates are not composed into a chain-level counterfactual.
ChainSWE [Jin et al., 2026]	Per-bug accuracy is 58.9% with oracle-correct prior state and 36.5% with agent-generated prior state.	Direct oracle-versus-persistent-state comparison.	The reported comparison does not compose local estimates into an expected chain outcome or test multiple mechanism-specific interventions.
SlopCodeBench [Orlanski et al., 2026]	No evaluated agent completes any of the 36 problems; erosion and verbosity increase in 77% and 75.5% of trajectories, respectively.	Agents repeatedly extend their own solutions.	The reported protocol characterizes quality degradation without a matched reset-state baseline.
SWE-Marathon [Desai et al., 2026]	Across 20 project-scale tasks and 1,300 rollouts, no configuration exceeds 30% pass@1; 0 of 71 compaction-path trials pass versus 8.9% without.	Persistent multi-hour rollouts with post-hoc trajectory audits.	Compaction and repetition correlate with failure but are not intervened on, so cause and marker remain confounded.
SWE-Milestone [Deng et al., 2026]	Independent milestone scores exceed roughly 80%; the best continuous score is 38.03%.	Canonical snapshots versus a persistent codebase.	The reported independent scores are not composed along semantic prerequisite paths.
Illusion [Sinha et al., 2026]	At a matched evaluation turn, increasing the error rate in a fixed-format prior history reduces subsequent accuracy.	Artificially healed or corrupted histories.	The controlled synthetic task does not establish prevalence or effect size in open-ended agents.
HORIZON [Wang et al., 2026]	More than 3,100 trajectories across four domains reveal distinct planning-, memory-, and history-related failure patterns.	Controlled depth or breadth extensions within task families.	Post-hoc, potentially co-occurring labels describe failure composition rather than causal mechanisms.

Table 4: Component-level delta between representative studies and the proposed protocol, judged from each study's reported protocol. ✓: implemented; ∼: partial or implicit; —: not part of the reported protocol.

Study	Matched local stages	Composed $P_{\texttt{expected}}$	Natural rollout	State compatibility	Targeted intervention	Protocol sensitivity
SWE-EVO	—	—	✓	—	—	—
NL2Repo-Bench	—	—	✓	—	—	—
SWE-Chain	∼	—	✓	—	—	—
ChainSWE	✓	—	✓	∼	∼	—
SlopCodeBench	—	—	✓	—	—	—
SWE-Marathon	—	—	✓	—	—	—
SWE-Milestone	✓	—	✓	∼	—	—
Illusion	✓	∼	∼	—	✓	—
HORIZON	—	—	✓	—	∼	—
This position (proposed)	✓	✓	✓	✓	✓	✓

Table 5: Evidence that interface design, agent scaffolding, environment state, and resource allocation are constitutive components of the evaluated system. Results follow study-specific protocols and are not directly comparable across rows.

Study	Design or reported signal	Relevance to this position
SWE-agent [Yang et al., 2024]	Custom agent-computer interface; 12.5% pass@1 on full SWE-bench.	Interface design changes natural interaction behavior and must be matched across conditions.
AutoCodeRover [Zhang et al., 2024]	AST-guided search and optional fault localization; 19% on SWE-bench Lite.	Search and localization support alter the effective local difficulty.
Agentless [Xia et al., 2024]	Fixed localization–repair–validation pipeline; 32.0% on SWE-bench Lite.	A controlled composition of stages can compete with open-ended trajectories.
MASAI [Arora et al., 2024]	Specialized sub-agents; 28.33% on SWE-bench Lite.	Modular decomposition changes context exposure and coordination burden.
WebArena [Zhou et al., 2023]	Realistic web workflows; 14.41% best-agent versus 78.24% human success.	End-to-end failure combines horizon with grounding, planning, and website state.
OSWorld [Xie et al., 2024]	369 desktop and web tasks; 12.24% best-model versus 72.36% human success.	Persistent operating-system state broadens the sources of trajectory failure.
AppWorld [Trivedi et al., 2024]	750 tasks across nine apps and 457 APIs, graded by state-based tests.	State checks can expose partial correctness and unintended collateral changes.
RE-Bench [Wijk et al., 2024]	Seven research-engineering environments with continuous scores and multiple budgets.	Capability conclusions depend on time allocation and best-of-k versus single-trajectory protocols.

Agent scaffolding as part of the evaluated system. The deployed agent is more than its base model. SWE-agent shows that an agent-computer interface can alter repository navigation, editing, and testing behavior [Yang et al., 2024]. AutoCodeRover instead uses program structure and fault localization to focus search [Zhang et al., 2024]; Agentless replaces open-ended tool planning with localization, repair, and patch validation [Xia et al., 2024]; and MASAI delegates reproduction, localization, repair, and ranking to specialized modules [Arora et al., 2024]. Their reported results use different models, prompts, costs, and benchmark versions, so they should not be read as a direct leaderboard. Collectively, they show why local-stage and natural-rollout evaluations must preserve the same agent configuration: changing the scaffold changes the capability being measured.

Stateful interactive environments beyond software repositories. The same concern appears outside software maintenance. WebArena evaluates realistic web workflows and reports 14.41% success for its best GPT-4-based agent versus 78.24% for humans [Zhou et al., 2023]. OSWorld evaluates 369 open-ended computer tasks and reports 12.24% for its best evaluated model versus 72.36% for humans [Xie et al., 2024]. AppWorld combines nine applications and 457 APIs with state-based tests that also detect collateral changes [Trivedi et al., 2024]. RE-Bench complements binary completion with continuously scored research-engineering objectives and varying time budgets; agents lead at short total budgets, while human experts show stronger returns to additional time [Wijk et al., 2024]. These benchmarks improve realism and stateful evaluation, but low success alone still mixes grounding, operational knowledge, planning, local difficulty, and horizon. Table 5 summarizes these system-level factors alongside the scaffolding studies above.

Partial-progress measurement and persistent-state controls. Terminal success can hide where a rollout failed. LHTB uses deterministic subtask grading, so runs with meaningful partial reward are distinguishable from both passing runs and no-progress failures [Li et al., 2026]. HORIZON analyzes more than 3,100 trajectories across web, operating-system, database, and embodied domains with a seven-category failure taxonomy [Wang et al., 2026]. Its trajectory-grounded judge reaches $\kappa = 0.84$ against one human annotator on a 40-trajectory pilot, supporting scalable descriptive annotation. The labels may co-occur and are assigned after observing the rollout, however, so agreement supports reproducibility of attribution rather than causal identification. SWE-Chain, ChainSWE, and SlopCodeBench instead preserve evolving software state across transitions or checkpoints [Lam et al., 2026, Jin et al., 2026, Orlanski et al., 2026]. Among the studies reviewed here, ChainSWE additionally reports a direct oracle-state comparison. These designs expose error propagation and architectural degradation that isolated issue protocols reset away, while leaving open how matched local performance should be composed into an expected end-to-end outcome.

Checkpoint composition and mechanism-targeted interventions. SWE-Milestone closely aligns with our proposed direction by comparing canonical-snapshot milestone execution with continuous execution on a persistent codebase [Deng et al., 2026]. Its inferred graph is useful for evaluating history and patch execution, but it need not be a minimal graph of semantic prerequisites: an alterna-

tive valid implementation may not reproduce the reference path. This illustrates why dependency annotations and state compatibility must be explicit when composing local estimates. More direct mechanism evidence comes from intervention. By manipulating errors in a fixed-format prior history, Sinha et al. [2026] estimate sensitivity to prior errors at a matched evaluation turn, while noting that the synthetic result is not sufficient for real-world long-horizon execution. Broader task ontologies and controlled horizon variation provide complementary ingredients [Dong et al., 2026, Kim et al., 2026]. Our position is that replayable checkpoints, dense progress measures, persistent-state comparisons, and targeted interventions should be combined under a pre-specified, agent-configuration-matched counterfactual, a combination that Table 4 shows no reviewed study yet implements in full.

7 Scope, Limitations, and Falsifiability

The proposed quantities are properties of a declared evaluation protocol, not intrinsic properties of a task or model. Stage decompositions are not unique, and dependency edges can be implementation-relative. Canonical checkpoints may also reveal privileged information about decomposition, localization, or the reference solution. We therefore advocate publishing the decomposition, acceptance conditions, dependency annotations, checkpoint construction, and revealed information, and testing whether conclusions persist across multiple reasonable annotations and both stage-given and goal-only conditions.

The decomposition is also a governance problem, not only a technical one. Someone must author it, and its quality bounds every downstream quantity. We recommend that decompositions be authored or reviewed by domain experts, versioned alongside the benchmark, and arbitrated through the annotation-sensitivity tests of Section 5 rather than by fiat; LLM-assisted decomposition is acceptable when each stage's acceptance condition is executed against ground truth, since an unverifiable stage silently redefines the estimand.

Statistical and resource assumptions are equally consequential. Local outcomes may be heterogeneous or correlated across tasks, stages, agents, and random seeds; separately started stages may receive more total search, verification, or retry budget than a natural rollout. Benchmark reports should therefore include raw counts, uncertainty, resource and cost accounting, conditional or simulation-based alternatives to the product model, and predeclared rules for zero and near-zero probabilities. Verifier error belongs in the same ledger: an agentic audit of graded rollouts reports judge–verifier disagreement of 1.4% for hand-written functional verifiers versus 32.4% for inherited pull-request tests [Huang et al., 2026]; at the latter rate, measurement noise alone can dominate a small residual, so residual estimates should report verifier validation alongside raw counts. Developing estimators that propagate uncertainty through branching state graphs is an important part of the empirical agenda.

A further risk is protocol sensitivity: two admissible checkpoint protocols can in principle yield opposite-signed residuals on the same task family. The protocol treats this scenario as a result rather than a failure. It indicates that no robust conclusion exists at the chosen resolution, but it also motivates the robustness criterion of Section 5: conclusions should be reported only when their sign is stable across the pre-declared admissible set, and as protocol ranges otherwise.

Finally, replayable and separately verifiable stages favor coding and terminal tasks with strong infrastructure. Task inclusion, public-repository exposure, reference-derived checkpoints, and verifier availability can introduce selection or contamination bias. The framework's value outside these domains remains an empirical question. Most importantly, Γ_{H} does not identify context degradation, planning failure, state contamination, or recovery; it identifies a contrast under one protocol.

Falsifiers of this position. Our position would be substantially weakened if any of the following were demonstrated:

1. across multiple pre-declared, admissible protocols on the same task family, the sign of Γ_{H} is unstable in ways that admissibility criteria cannot resolve;
2. compatible checkpoints cannot be constructed at manageable cost for representative coding benchmarks; or
3. the residual provides no incremental diagnostic information beyond existing persistent-versus-reset comparisons across several benchmark families.

We invite empirical work targeting these tests.

8 Anticipated Objections and Research Agenda

Q: Isn't end-to-end success what deployment cares about? **A:** Yes. Deployment and diagnostic benchmarks answer different questions, and neither replaces the other. Our claim applies only when a deployment result is used to argue for a particular failure mechanism.

Q: Won't scaling dissolve the problem? **A:** Longer contexts and better base models may reduce the residual, but they do not directly address state contamination, planning, or recovery. If a scaled system reaches $\Gamma_{\mathrm{H}} \approx 0$ under a declared baseline, that is exactly the evidence this position asks for.

Q: Don't standard ablations already identify mechanisms? **A:** Ablations measure component contributions within a fixed protocol. The residual asks whether matched local performance predicts the end-to-end result. The two are complementary: the residual locates a mismatch that an ablation can then help explain.

Q: If everything depends on the decomposition, how can results be compared? **A:** Publish, replicate, and vary the decomposition. The robustness criterion in Section 5 withholds conclusions that do not survive reasonable alternatives. Declared dependence can be audited; hidden dependence cannot.

The objections above mark the boundaries of the claim; the following agenda moves within them:

1. **Instantiate the comparison.** Apply the protocol to one existing persistent-state benchmark and report P_{expected}, P_{observed}, and Γ_{H} with bootstrap intervals, reusing published decompositions where available.
2. **Develop branching estimators.** Extend the uncertainty treatment of Equation 4 to conditional transition models over branching state graphs, with explicit rules for zero and near-zero probabilities.
3. **Run the intervention program.** Pair each recurring residual pattern with single-factor interventions, history reset or compression, state repair, verification, rollback, explicit planning, so that residual patterns become testable causal hypotheses rather than labels.
4. **Map protocol robustness.** Characterize when admissible protocols agree in sign and when they diverge, turning the robustness criterion of Section 5 into empirically grounded admissibility rules.
5. **Transfer beyond coding.** Identify where replayable state and semantic acceptance conditions exist outside coding and terminal tasks, and adapt the protocol where they do not.

We close with the strongest form of the claim. A declining end-to-end curve licenses a claim about deployment; a claim about long-horizon *mechanisms* should be licensed only by an evaluation that reports, alongside end-to-end success, a pre-specified compositional prediction and the resulting residual.

9 Conclusion

Long tasks are globally harder in part because they require more work. Our position is that the central scientific question is whether natural end-to-end rollouts perform differently from a pre-specified, intervention-specific model of local competence. We therefore advocate pre-specifying the stage decomposition and intervention protocol, estimating a protocol-specific counterfactual, measuring the natural rollout, and using targeted interventions to investigate the resulting contrast. The horizon residual is not a causal answer; it is a disciplined starting point for testing which aspects of history, state, information, and recovery make long-horizon execution differ. Until such residuals are reported, a declining curve should be read as evidence about deployment, not about mechanism. This framing turns "longer is harder" from a descriptive observation into an auditable research program.

References

Daman Arora, Atharv Sonwane, Nalin Wadhwa, Abhav Mehrotra, Saiteja Utpala, Ramakrishna Bairi, Aditya Kanade, and Nagarajan Natarajan. MASAI: Modular architecture for software-engineering AI agents. *arXiv preprint arXiv:2406.11638*, 2024. URL https://arxiv.org/abs/2406.11638.

Gangda Deng, Zhaoling Chen, Zhongming Yu, Haoyang Fan, Yuhong Liu, Yuxin Yang, Dhruv Parikh, Rajgopal Kannan, Le Cong, Mengdi Wang, Qian Zhang, Viktor Prasanna, Xiangru Tang, and Xingyao Wang. SWE-Milestone: Evaluating AI agents on continuous software evolution. *arXiv preprint arXiv:2603.13428*, 2026.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025. URL https://arxiv.org/abs/2509.16941.

Rishi Desai, Jesse Hu, Joan Cabezas, Neel Harsola, Pratyush Shukla, et al. SWE-Marathon: Can agents autonomously complete ultra-long-horizon software work? *arXiv preprint arXiv:2606.07682*, 2026. URL https://arxiv.org/abs/2606.07682.

Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, Chao He, Yue Hou, Fei Hu, Zhaojian Li, Weiran Shi, Zaiyuan Wang, Daoguang Zan, Chenchen Zhang, Xiaoxu Zhang, Qizhi Chen, Xianfu Cheng, Bo Deng, Qingshui Gu, Kai Hua, Juntao Lin, Pai Liu, Mingchen Li, Xuanguang Pan, Zifan Peng, Yujia Qin, Yong Shan, Zhewen Tan, Weihao Xie, Zihan Wang, Yishuo Yuan, Jiayu Zhang, Enduo Zhao, Yunfei Zhao, He Zhu, Liya Zhu, Chenyang Zou, Ming Ding, Jianpeng Jiao, Jiaheng Liu, Minghao Liu, Qian Liu, Chongyang Tao, Jian Yang, Tong Yang, Zhaoxiang Zhang, Xinjie Chen, Wenhao Huang, and Ge Zhang. NL2Repo-Bench: Towards long-horizon repository generation evaluation of coding agents. *arXiv preprint arXiv:2512.12730*, 2025. URL https://arxiv.org/abs/2512.12730.

Guanting Dong, Xiaoshuai Song, Yuyang Hu, Jiajie Jin, Chenghao Zhang, Yifei Chen, Xiaoxi Li, Huaying Yuan, Xinyu Yang, Tongyu Wen, Jiejun Tan, Hongjin Qian, Shijue Huang, Junting Lu, Zhenyu Li, Wanjun Zhong, Yutao Zhu, Tat-Seng Chua, Zhicheng Dou, and Ji-Rong Wen. Towards long-horizon agents: A survey—foundation, evolution, harness, optimization, application, and frontier. *Preprint*, 2026. URL https://long-horizon-agents.github.io.

Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Technical report, Chroma, 2025. URL https://research.trychroma.com/context-rot.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*, 2026. URL https://arxiv.org/abs/2607.07946.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770. arXiv:2310.06770.

Qirui Jin, Lingching Tung, Kenan Li, Qiyang Shi, Yushi She, Huanzhong Jia, Harrison Zhao, Kejing Xia, Zhenbang Du, Yikai Zhang, Jiaxin Pei, Zhenyu Zhang, Zhen Qi, Yuyan Duan, Wenke Lee, and Zijian Jin. ChainSWE: Benchmarking coding agents on multi-bug software maintenance. *arXiv preprint arXiv:2607.02606*, 2026. URL https://arxiv.org/abs/2607.02606.

Sunghwan Kim, Junhee Cho, Beong-woo Kwak, Taeyoon Kwon, Liang Wang, Nan Yang, Xingxing Zhang, Furu Wei, and Jinyoung Yeo. On training large language models for long-horizon tasks: An empirical study of horizon length. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026. arXiv:2605.02572.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Chris Painter, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long software tasks. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2503.14499. arXiv:2503.14499.

Man Ho Lam, Chaozheng Wang, Hange Liu, Jingyu Xiao, Haau-sing Li, Jen-tse Huang, Terry Yue Zhuo, and Michael R. Lyu. SWE-Chain: Benchmarking coding agents on chained release-level package upgrades. *arXiv preprint arXiv:2605.14415*, 2026. URL https://arxiv.org/abs/2605.14415.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. SWE-EVO: Benchmarking coding agents in long-horizon software evolution scenarios. *arXiv preprint arXiv:2512.18470*, 2025. URL https://arxiv.org/abs/2512.18470.

Zongxia Li, Zhongzhi Li, Yucheng Shi, Ruhan Wang, Junyao Yang, Zhichao Liu, Xiyang Wu, Anhao Li, Yue Yu, Ninghao Liu, Lichao Sun, Haotao Mi, and Leowei Liang. 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.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. arXiv:2305.20050.

Jiayuan Liu, Tianqin Li, Shiyi Du, Xin Luo, Haoxuan Zeng, Emanuel Tewolde, Tai Sing Lee, Tonghan Wang, Carl Kingsford, and Vincent Conitzer. The memory curse: How expanded recall erodes cooperative intent in LLM agents. *arXiv preprint arXiv:2605.08060*, 2026. URL https://arxiv.org/abs/2605.08060.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-Lancer: Can frontier LLMs earn $1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*, 2025. URL https://arxiv.org/abs/2502.12115.

Gabriel Orlanski, Devjeet Roy, Alexander Yun, Changho Shin, Alex Gu, Albert Ge, Dyah Adila, Nicholas Roberts, Frederic Sala, and Aws Albarghouthi. SlopCodeBench: Benchmarking how coding agents degrade over long-horizon iterative tasks. *arXiv preprint arXiv:2603.24755*, 2026. URL https://arxiv.org/abs/2603.24755.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2412.21139. arXiv:2412.21139.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in LLMs. In *International Conference on Learning Representations*, 2026. arXiv:2509.09677.

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*, 2024. URL https://arxiv.org/abs/2407.18901. arXiv:2407.18901.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers

as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2407.16741. arXiv:2407.16741.

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.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024. URL https://arxiv.org/abs/2411.15114.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024. URL https://arxiv.org/abs/2407.01489.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793. arXiv:2405.15793.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024. URL https://arxiv.org/abs/2404.05427. arXiv:2404.05427.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.