Building Reliable Long-Horizon Agents: A Survey

Definitions, Metrics, Benchmarks, and System Design

Kai Wu[1,*], Hao Lyu[1,*], Zhen Luo[1,*], Chaofan Wang[2], Siyu Ye[6], Jinghao Lin[6], Xiaozhong Ji[3], Boyuan Jiang[4]
Yiwen Ye[6], Zimu Wang[5], Wenzhe Liu[6], Ruobing Wang[6], Kai Cai[6], Shengzhi Wang[1], Qingwen Liu[1,†]

[*]Equal contribution. [†]Corresponding author.

[1]Tongji University; [2]Shanghai Jiao Tong University; [3]Nanjing University;
[4]Zhejiang University; [5]University of California, Berkeley; [6]Simple Agent Lab

LLM agents are shifting from producing isolated answers to executing long-running, stateful tasks across software, web, computer, and scientific environments. This transition makes long-horizon execution a central frontier because errors can change external state and shape later decisions. Yet *long horizon* remains defined inconsistently through context length, elapsed time, action count, tool calls, or benchmark difficulty. We define a long-horizon task by cross-step coupling. Earlier actions create information, state, constraints, or consequences that later decisions must preserve and verify. We organize long-horizon execution metrics around six task-pressure axes: human-effort time, interaction length, context and memory demand, environment grounding, planning dependency, and verification observability. A protocol-conditioned outcome stack measures task success, consistency, safety, progress, cost, and uncertainty. This separation reveals the central bottleneck: progress depends not only on peak capability, but on whether execution remains reliable as task pressure increases.

Drawing on 201 cited research papers and technical sources, we synthesize the capabilities that determine this reliable horizon. The survey is organized around a *model–harness–environment–evaluation* stack. Model design shapes reasoning, planning, tool use, and trajectory learning. Harness design governs context, memory, feedback, verification, and oversight. Environments determine state transitions, grounding demands, and observability, while evaluation protocols determine which reliability claims are justified. This systems view explains how the four layers jointly enable or constrain reliable long-horizon execution. We identify open challenges in comparable stress paths, replayable environments, delayed verification, model–harness attribution, contamination, cost, and safe autonomy. The resulting framework provides a roadmap for extending agent horizons without sacrificing reliability, safety, or reproducibility.

Main Contact:
Github:

Figure 1: Conceptual overview of the research landscape for reliable long-horizon agents, spanning five benchmark domains (Section 3), model design (Section 4), harness design (Section 5), and open problems beyond the current reliable horizon (Section 6). Logos indicate representative systems and are illustrative rather than exhaustive.

Contents

1 Introduction

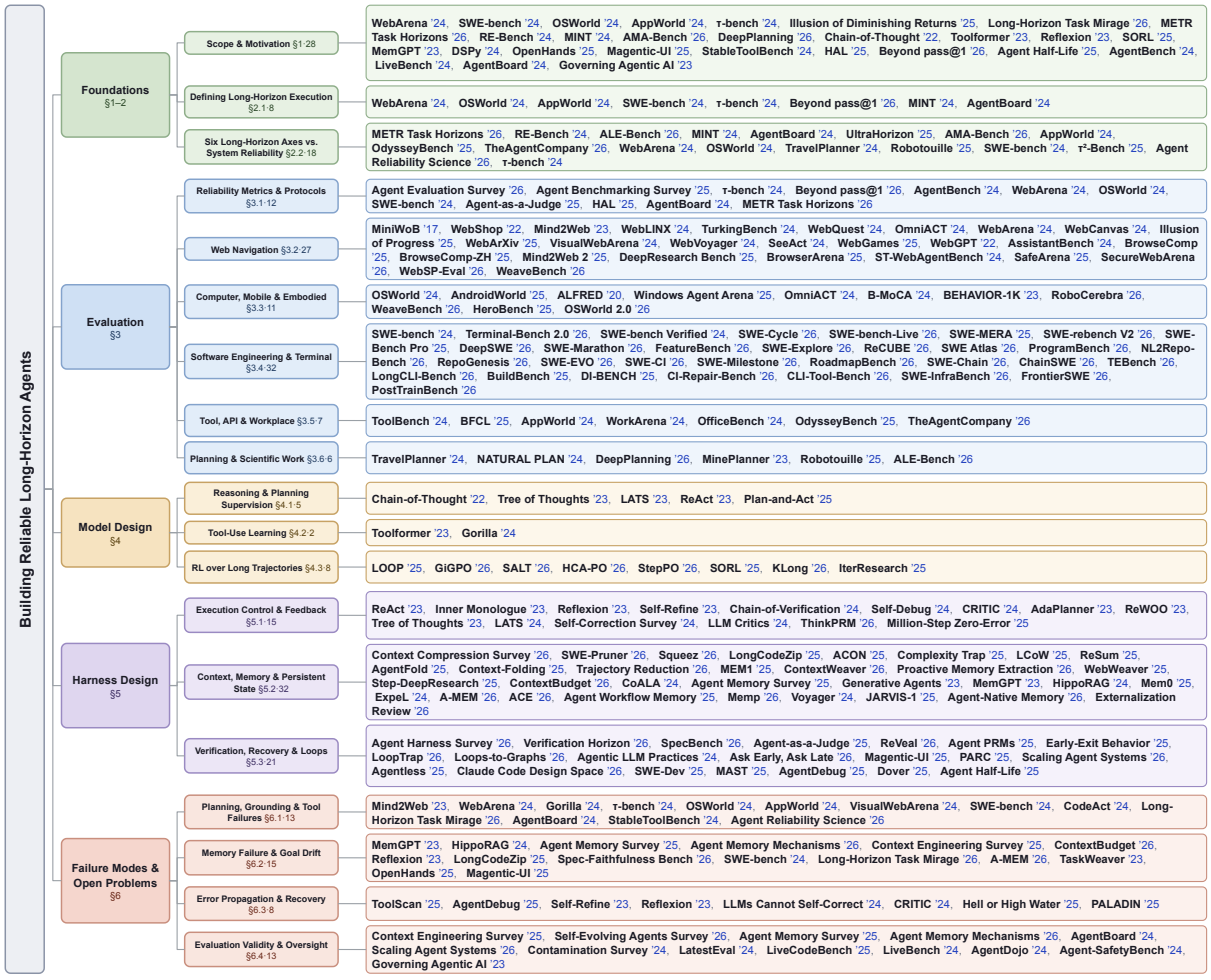

Figure 2: Survey map linking each section to its long-horizon themes and representative evidence. Branches organize the literature; years identify source records rather than comparable performance values.

Long-horizon tasks have become a defining frontier of agent development: leading model developers now position Claude Fable 5, GPT-5.6, Kimi K3, and GLM-5.2 for extended coding, research, computer use, and other sustained agentic work.[1] This convergence marks a decisive shift in frontier-agent research from producing isolated answers toward completing long-running, stateful tasks. Figure 1 previews the benchmark and systems landscape considered in this survey. Rather than emitting a single response to a static input, agents are increasingly expected to browse websites (Zhou et al., 2024b), edit code repositories (Jimenez et al., 2024), operate graphical interfaces (Xie et al., 2024b), call APIs and query databases (Trivedi et al., 2024a), and coordinate tool-agent-user workflows (Yao et al., 2024). In these settings, the basic unit of behavior is an interactive trajectory in which actions alter external state and condition subsequent observations, decisions, and success criteria. The four newly released systems are therefore used here as evidence of the direction of industrial investment, not as commensurate proof that one system has a longer reliable horizon: their public results use different harnesses, budgets, tasks, and graders.

Despite rapid progress, the notion of *long-horizon* agency remains under-specified. Existing work often associates horizon with long contexts, many interaction turns, prolonged wall-clock duration, large numbers of tool calls, or difficult benchmark instances. These proxies are useful but insufficient: an

[1] Official product sources: Anthropic: Claude Fable 5, OpenAI: GPT-5.6, Moonshot AI: Kimi K3, and Z.ai: GLM-5.2. Accessed 21 July 2026.

agent may process a long document without performing long-horizon execution, while a comparatively short interaction may become long-horizon when early actions create constraints that later actions must respect (Sinha et al., 2025; Wang et al., 2026c). We therefore define horizon as the extent over which an agent must preserve goal-relevant information, maintain a valid representation of the world state, select dependent actions, and verify delayed consequences. This view separates horizon from surface length and instead emphasizes dependency depth, mutable state, memory persistence, grounding, delayed verification, and repeated-run reliability. METR-style long-task evaluation captures one dimension of horizon through the human time required for tasks that systems can complete at a specified success rate (Kwa et al., 2026), while τ-bench captures another by evaluating whether tool-using agents can succeed consistently across repeated trials rather than only once (Yao et al., 2024).

In this survey, we argue that long-horizon LLM agents should be studied as coupled *model–harness–environment–evaluation* systems. Figure 2 maps this scope to the paper's sections and representative evidence. Models determine local transition quality by decomposing goals and reasoning over intermediate steps (Wei et al., 2022), grounding actions through tool-use learning (Schick et al., 2023), revising after feedback (Shinn et al., 2023), and learning from delayed rewards in agentic settings (Li et al., 2025a). However, model capability alone does not determine the effective horizon of an agent. The surrounding harness—including memory management (Packer et al., 2023), programmatic prompt and module composition (Khattab et al., 2024), software-agent execution environments (Wang et al., 2025d), and mixed-initiative human oversight (Mozannar et al., 2025)—shapes whether local errors are prevented, detected, isolated, or allowed to propagate. Evaluation protocols further determine whether observed improvements reflect genuine reliable autonomy or artifacts of sampling budget, scaffolding, benchmark contamination, or unstable environments (Guo et al., 2024; Kapoor et al., 2025).

Scope. We include interactive LLM agents whose success depends on coupled actions, changing state, delayed effects, or trajectory-level checks. We exclude ordinary long-context processing, independent dialogue turns, and one-call tool use unless they test horizon-dependent reliability.

Research questions and contributions

Research questions

Q1	What makes agent execution long-horizon?	[§2]
Q2	How should its reliable horizon be measured?	[§3]
Q3	How can model and harness design extend it?	[§§4–6]

Contributions

C1	A unified definition and measurement framework for reliable horizon.	[§2]
C2	An evidence-coded map of 64 long-horizon benchmarks.	[§3]
C3	A model–harness taxonomy for extending reliable execution.	[§§4–5]
C4	A proof checklist for credible reliable-horizon claims.	[§§3, 6]

We organize existing work through the lens of the *reliable execution horizon*: the range over which an agent can continue acting while remaining correct, grounded, recoverable, and cost-effective. This construct explains why long-horizon settings are qualitatively different from single-step prediction: small local weaknesses can compound when errors enter memory, corrupt external state, invalidate future preconditions, or mislead subsequent verification (Ord, 2025; Khanal et al., 2026). Accordingly, the main text follows six stages. We first define long-horizon execution and formulate reliable horizon over a multidimensional task-pressure space. We then examine the benchmarks, metrics, and protocols used to measure it. Next, we review model design and harness design as two complementary routes to more reliable execution. We then analyze recurring failure modes and open problems before drawing the system-level implications. Appendix A documents the search, screening, extraction, and source-verification process. Across the survey, we use benchmarks such as AgentBench, LiveBench, WebArena, OSWorld, SWE-bench, AppWorld, and τ-bench (Liu et al., 2024; White et al., 2024; Zhou et al., 2024b;

Xie et al., 2024b; Jimenez et al., 2024; Trivedi et al., 2024a; Yao et al., 2024), together with protocols for reliability, attribution, cost, contamination, and reproducibility (Ma et al., 2024; Shavit et al., 2023). We conclude that progress requires measuring and extending reliable execution horizon as a property of the full model–harness–environment–evaluation stack.

2 Foundations: Defining and Measuring Reliable Horizon

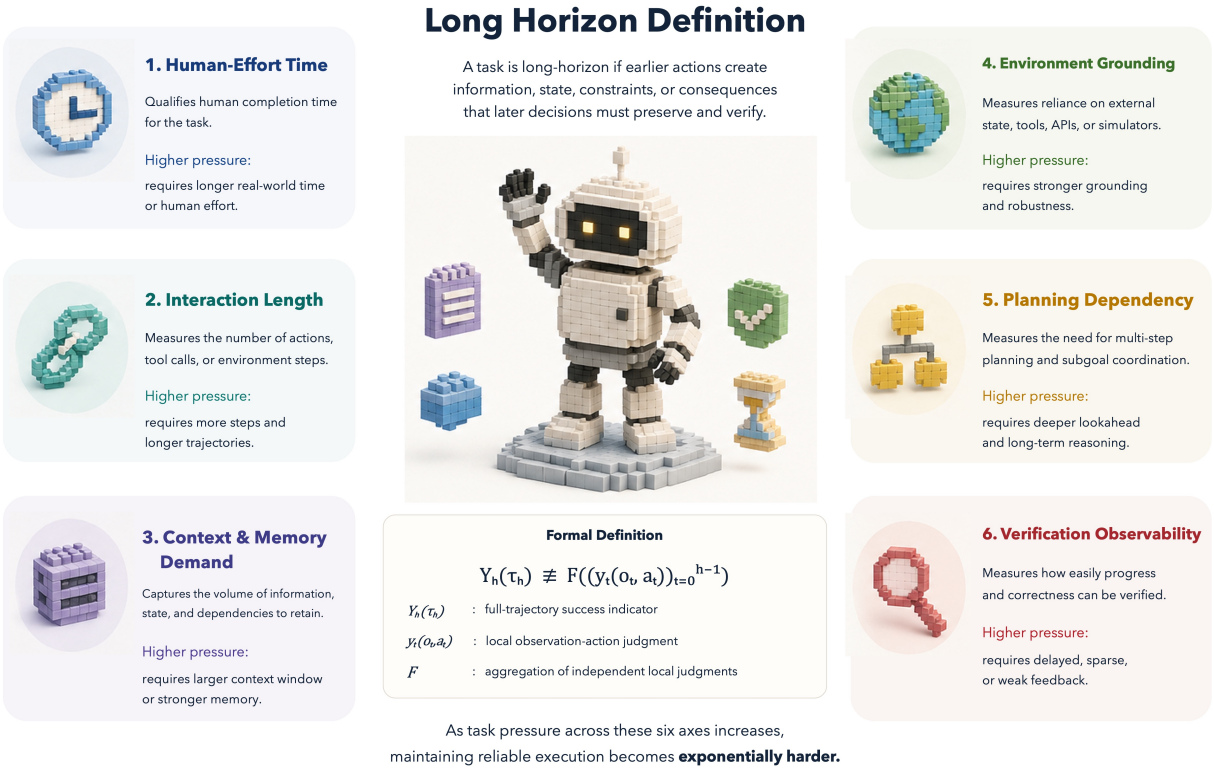

Figure 3: Conceptual overview of the long-horizon definition and its six task-pressure axes. Cross-step coupling distinguishes long-horizon execution from merely long inputs or independent steps: earlier actions create information, state, constraints, or consequences that later decisions must preserve and verify. Human-effort time, interaction length, context and memory demand, environment grounding, planning dependency, and verification observability characterize how task pressure increases; reliability is the response of a fixed system under a fixed protocol.

This section separates three questions that are often conflated. First, what makes an execution genuinely long-horizon? Second, how is it long-horizon? Third, how reliably does a fixed system perform as task pressure increases? Section 2.1 defines long-horizon execution through cross-step coupling. Section 2.2 then refines this definition with six task-pressure axes and measures system reliability over them.

2.1 Defining Long-Horizon Agent Execution

Across recent agent benchmarks, the evaluation unit is no longer an isolated response but an execution trajectory. Web and GUI tasks require actions to remain grounded as interfaces change (Zhou et al., 2024b; Xie et al., 2024b). App, tool, and software tasks require state changes to satisfy executable checks (Trivedi et al., 2024a; Jimenez et al., 2024). Repeated-trial protocols further ask whether success is reproducible rather than accidental (Yao et al., 2024; Khanal et al., 2026). Across these settings, earlier actions change the information, state, or constraints that later decisions must respect.

We therefore define *long-horizon execution* by cross-step coupling, not by raw length. An execution is long-horizon when success depends on preserving goal-relevant information, tracking mutable state, selecting dependent actions, incorporating feedback, and verifying delayed consequences. Prompt length,

wall-clock duration, turn count, and tool-call count may correlate with this structure, but none is sufficient on its own.

Let an agent policy π_θ interact with an environment $\mathcal{E}$ over horizon length h, producing a trajectory

$$\begin{aligned} \tau_h &= (x_0, o_0, a_0, \ldots, x_h), \\ a_t &\sim \pi_\theta(\cdot \mid c_t), \\ x_{t+1} &\sim P_{\mathcal{E}}(\cdot \mid x_t, a_t), \qquad t = 0, \ldots, h-1. \end{aligned}$$

Here x_t is the environment state, o_t is the observation, a_t is the action, and c_t is the context or memory used for action selection.

The defining boundary is a compression test. If a task can be reduced to independent one-step predictions without changing its success condition, it is multi-step but not strongly long-horizon. The task enters the long-horizon regime when that reduction removes information needed for later decisions or final verification.

Formally, let $Y_h(\tau_h) \in \{0, 1\}$ denote whether the full trajectory succeeds. A strongly long-horizon task is one for which success is not reducible to independent local judgments:

$$Y_h(\tau_h) \not\equiv F\Big((y_t(o_t, a_t))_{t=0}^{h-1}\Big).$$

Here y_t is a local observation–action judgment that omits cross-step dependencies. The expression is a diagnostic rather than a claim that every long trajectory is difficult. It asks whether local judgments retain enough information to determine full-trajectory success. MINT, AgentBoard, AppWorld, and τ-bench operationalize this distinction through interaction, progress, state change, or repeated trials (Wang et al., 2024d; Ma et al., 2024; Trivedi et al., 2024a; Yao et al., 2024).

This definition excludes common false positives. A long document is not sufficient if no action changes external state. Many tool calls are not sufficient if they are independent. Conversely, a short trajectory can be strongly long-horizon when an early action changes later feasible actions or delays the only useful correctness signal. The definition supplies a structural criterion, not a universal scalar length.

The definition answers whether an execution is long-horizon, but not how it is long-horizon. Section 2.2 answers the latter question with six task-pressure axes. Reliability remains a system outcome, not a seventh task axis.

2.2 Six Long-Horizon Axes vs. System Reliability

The cross-step coupling test in Section 2.1 determines whether an execution is long-horizon. It does not explain how the task is long-horizon. We refine the definition with six task-pressure axes: human-effort time, interaction length, context and memory demand, environment grounding, planning dependency, and verification observability. Each axis identifies one way in which cross-step coupling becomes harder to sustain. A task may be long along one axis or several axes at once. These axes describe the task, whereas reliability measures a fixed system under a fixed protocol. Let

$$\begin{aligned} \mathbf{z} = (&z_{\text{time}}, z_{\text{interaction}}, z_{\text{context}}, \\ &z_{\text{grounding}}, z_{\text{dependency}}, z_{\text{observability}}) \in \mathcal{Z} \end{aligned}$$

denote the task-pressure vector. Its coordinates are human-effort time, interaction length, context and memory demand, environment grounding, planning dependency, and verification observability. For an agent system $S = (\pi_\theta, \mathcal{G}, \mathcal{E})$, let $\mathcal{G}$ denote the harness. Let protocol Q fix budgets, retries, tools, graders, and sampling conditions. We define

$$R_{S,Q}(\mathbf{z}) = \Pr_{\tau \sim (S,Q) \mid \mathbf{Z} = \mathbf{z}}\big[Y_Q(\tau) = 1\big]. \tag{1}$$

Here Y_Q is trajectory-level success under the state, safety, and verification checks declared by Q. The reliable region is

$$\mathcal{H}_\alpha(S, Q) = \big\{\mathbf{z} \in \mathcal{Z} : R_{S,Q}(\mathbf{z}) \geq \alpha\big\}, \tag{2}$$

and its boundary is the *reliable-horizon surface*. The formulation does not make the six coordinates interchangeable.

A one-dimensional horizon is valid only after declaring how task pressure increases. For a non-decreasing stress path $g : h \mapsto \mathbf{z}$, define

$$H_\alpha(S, Q; g) = \sup\{h : R_{S,Q}(g(h)) \geq \alpha\}. \tag{3}$$

When all other coordinates are fixed, the corresponding conditional horizon along axis d is

$$H_\alpha^{(d)}(\mathbf{z}_{-d}; S, Q) = \sup\{u : R_{S,Q}(\mathbf{z}_{-d}, z_d = u) \geq \alpha\}. \tag{4}$$

Equations 3 and 4 have a threshold-crossing interpretation only when reliability is non-increasing along the chosen path. For a non-monotone path, we report the reliable set or use the robust prefix

$$H_\alpha^{\text{prefix}}(S, Q; g) = \sup\Big\{h : \inf_{0 \leq u \leq h} R_{S,Q}(g(u)) \geq \alpha\Big\}, \tag{5}$$

which prevents an isolated easier instance at a larger nominal h from inflating the horizon.

Table 1 summarizes the operational meaning of the six axes and the corresponding reliability question. The axes are inputs to the reliability surface in Equation (1). Task success, repeated-run consistency, safety, progress, cost, and uncertainty form the outcome stack. This distinction prevents a task property from being confused with evidence about the agent.

Table 1: Six long-horizon axes and their reliability interpretation. The first two columns define task pressure. The third states the system response to estimate under fixed S and Q.

Long-horizon axis	Task pressure	Reliability question
Human-effort time	Qualified-human completion time for the task	Does reliability remain above α in longer human-time bins?
Interaction length	Dependent actions, tool calls, GUI operations, or state transitions	How quickly do success and recovery decline as dependency length increases?
Context and memory demand	Goal-relevant state that must remain available, current, and usable	Does reliability survive greater state volume, update frequency, and stale-state exposure?
Environment grounding	Sustained alignment among intent, observation, and executable action	Does reliability persist as interfaces and environment states change?
Planning dependency	Global constraints, ordering, resource coupling, and delayed consequences	Does reliability persist as constraint density and dependency depth increase?
Verification observability	Delay, sparsity, and localizability of correctness signals	Can the system detect and recover as feedback becomes delayed or sparse?

Human-Effort Time Axis. The human-effort time axis measures task extent by the time a qualified human requires for completion. METR-style evaluations report the human completion time that an AI system can match at a declared success threshold (Kwa et al., 2026). This coordinate provides an interpretable unit for otherwise heterogeneous task suites.

This axis is useful for software engineering, AI R&D, and algorithm engineering. RE-Bench compares agents with expert humans in multi-hour ML research environments (Wijk et al., 2024). ALE-Bench evaluates algorithm engineering through repeated submission, scoring, visualization, and refinement (Imajuku et al., 2026). In both cases, human labor time provides the extent coordinate, while other axes explain why the task is difficult.

Reliability analysis. Estimate $R_{S,Q}$ within human-time bins while holding the protocol fixed. A claimed extension requires reliability to remain above α at longer durations. It should not result only from a higher mean score on a different task mixture.

Interaction-Length Axis. The interaction-length axis measures the number and dependency structure of actions, tool calls, GUI operations, or environment transitions. MINT shows that multi-turn tool use and language feedback create demands that single-turn evaluation does not expose (Wang et al., 2024d). AgentBoard complements binary success with progress rate, revealing how much of a task was completed before success or failure (Ma et al., 2024).

UltraHorizon extends this coordinate with partially observable exploration tasks that reach hundreds of thousands of tokens and hundreds of tool calls (Luo et al., 2025). The resulting pressure comes from both interaction count and interaction structure. Agents must maintain hypotheses, explore systematically, and recover from early misleading patterns.

Reliability analysis. Condition reliability on dependent interaction count or dependency depth, not raw action count alone. Report success, progress, and recovery curves as the interaction structure grows. Independent repeated calls should not be treated as equivalent to cascaded decisions.

Context and Memory-Demand Axis. The context and memory-demand axis measures how much goal-relevant state must remain available and correct. This state includes user constraints, artifacts, tool outputs, file edits, database state, and prior observations. Unlike ordinary long-context inputs, these representations arise from earlier actions and affect later state transitions.

AMA-Bench evaluates memory in agentic rather than dialogue-only settings (Zhao et al., 2026b). AppWorld (Trivedi et al., 2024a), OdysseyBench (Wang et al., 2025c), and TheAgentCompany (Xu et al., 2026a) also require persistent cross-application state. Agents must retain actionable information, replace stale state, and reconstruct the current task state.

Reliability analysis. Stratify reliability by state volume, update frequency, and stale-state exposure. Oracle memory ablations help separate memory failures from task difficulty.

Environment-Grounding Axis. The environment-grounding axis measures the pressure created by maintaining perception–action alignment in a changing environment. WebArena uses realistic self-hosted websites and evaluates functional correctness rather than surface-form action matching (Zhou et al., 2024b). OSWorld extends this setting across GUIs, CLIs, file systems, and multi-application workflows (Xie et al., 2024b).

Every action can change the future observation stream. A wrong click, incorrect file edit, or unintended window switch may corrupt state and mislead subsequent decisions. Grounding pressure therefore concerns sustained alignment among intent, observed state, and executable actions, not single-step perception accuracy.

Reliability analysis. Measure final-state success together with invalid-action rate and recovery after grounding errors. Comparisons should vary interface or state volatility under the same protocol. Terminal success alone can hide transient errors and costly recovery.

Planning-Dependency Axis. The planning-dependency axis measures the depth of global constraints, temporal dependencies, resource limits, and delayed consequences. TravelPlanner requires agents to synthesize database evidence into plans that satisfy budget, route, lodging, cuisine, and commonsense constraints (Xie et al., 2024a). Difficulty persists even when relevant information is supplied, which isolates global dependency management from tool use.

Local progress does not guarantee global coherence along this axis. An agent may complete plausible subtasks while violating budgets, temporal consistency, route feasibility, tests, or software specifications. Robotouille (Gonzalez-Pumariega et al., 2025) and SWE-bench (Jimenez et al., 2024) expose this pressure because early decisions restrict later solutions and local fixes can introduce global failures.

Reliability analysis. Stratify results by dependency depth and constraint density. Report final success, constraint violations, and recovery from invalid partial plans. Longer plan text is not evidence of reliability. Solver or oracle-plan ablations can isolate global coordination failures.

Verification-Observability Axis. The verification-observability axis measures how late, sparse, and localizable correctness signals are. At one extreme, every action produces an immediate executable check. At the other, correctness appears only through a final state, delayed user feedback, or a partly subjective

judgment. Delayed or weak signals allow more downstream actions to depend on undetected faults and make rollback or attribution harder.

SWE-bench exposes delayed but executable evidence through issue-specific and regression tests (Jimenez et al., 2024). AppWorld and OSWorld judge state changes with unit tests or execution-based checks (Trivedi et al., 2024a; Xie et al., 2024b). τ^2-Bench adds user-side actions whose correctness may not be controlled directly by the agent (Barres et al., 2025). These benchmarks differ in when and how errors become observable. Observability is the task pressure; reliability is the measured outcome.

Reliability analysis. Vary feedback delay and sparsity while preserving the underlying task logic. Report detection latency, rollback success, escaped-error rate, and final success. This analysis tests whether the system remains dependable when verification arrives late.

Realistic tasks often apply pressure along several axes. OSWorld combines interaction length, environment grounding, context persistence, and delayed execution-based verification. TravelPlanner emphasizes planning dependency but also requires tool interaction and final constraint checks. Long software tasks use human-effort time as an interpretable coordinate, while failures may arise from memory, testing feedback, tool use, cross-file dependencies, or delayed regressions.

The six-axis taxonomy is therefore a coordinate system, not a partition of benchmarks. A benchmark may stress several axes, and a system may fail only under particular combinations. Evaluation estimates the outcome stack: task success, repeated-run consistency, process validity, safety, efficiency, and uncertainty (Rabanser et al., 2026; Yao et al., 2024). Cross-step coupling determines whether execution is long-horizon. The six axes specify how it is long-horizon, and the reliability surface measures how well a fixed system tolerates that pressure. This input–response separation organizes the rest of the survey. Benchmarks specify where pressure is applied, model and harness methods intervene on the system, and evaluation tests whether the reliable-horizon surface moves outward.

3 Evaluation: Benchmarks, Metrics, and Protocols

This section maps long-horizon agent benchmarks to the real-world settings in which reliable execution is required. Rather than treating benchmarks as a flat list, we organize them by the dominant environment and failure mode they expose: web navigation and browsing, computer and mobile use, software engineering and terminal work, tool/API/app workflows, and constraint-heavy planning or scientific work. Across these domains, the central evaluation target is not only final task success, but whether an agent can preserve state, coordinate dependent actions, ground decisions in changing environments, and remain reliable across repeated executions. Table 2 compares representative anchor benchmarks using a typed, within-modality extent level, the official per-task evidence behind that level, and the concrete mechanisms through which actions remain coupled, state persists, and correctness becomes observable. The complete 64-entry inventory appears in Appendix B.

A scalar horizon is meaningful within a benchmark family only when instances form a declared, approximately nested stress path and the system and protocol are held fixed. In WebArena, such a path might increase interaction depth or cross-site grounding while preserving the task family; in SWE-bench, it might increase qualified-human time or cross-file dependency under the same test protocol; and in AppWorld, it might increase dependent API calls, persistent-state breadth, or verification delay. These are different paths through the six-dimensional space, so their numerical horizons are not directly comparable and should not be pooled under an unqualified $\max h$. When success is non-monotone along a proposed path, evaluations should report the reliable set or the robust-prefix horizon in Equation (5), not select the largest isolated successful instance.

3.1 Reliability Metrics and Experimental Protocols

Language-agent evaluation should be organized as a metric stack rather than a single success score. Unlike static language-model benchmarks, agent benchmarks involve stochastic rollouts, tool calls, state transitions, intermediate actions, environment feedback, and resource constraints. Recent surveys therefore frame agent evaluation as a multi-dimensional problem covering capabilities, behavior, reliability, safety, interaction modes, metric computation, and evaluation infrastructure (Yehudai et al., 2026; Mohammadi

Table 2: Representative long-horizon agent benchmarks, typed within-modality extent, and concrete horizon evidence. Benchmark names link to their source references.

Benchmark	Domain	Scale	Type	Per-task extent	Coupling	Persistent state	Correctness signal
WebShop	Web Shopping	12,087 instructions	S2	11.3 states (mean; human expert); max 114	Product constraints couple search, comparison, option selection, and purchase	Visited products, selected options, cart, and session state	Task reward; exact success when reward equals 1
WebArena	Web Navigation	812 tasks	T1	35.38 s (median; external 100-task sample)	Cross-page and cross-site subgoals	Website, account, and database state	Task-specific functional checks over final state or answer
OSWorld	Desktop Computer Use	369 tasks	T2	111.94 s (median; $n = 369$)	UI and file operations create later preconditions	Files, applications, documents, settings, and OS state	Execution-based final application, file, and OS checks
OSWorld 2.0	Professional Computer Workflows	108 tasks	T4	About 1.6 h (median; annotated/estimated); 69.6% $>$ 1 h	Interdependent cross-application workflows with dynamic updates	Workspace files, application/service records, user-profile data, messages, and injected updates	Fine-grained final-state checkpoints; functional checks with limited model-based judging
ALFRED	Embodied Household Tasks	8,055 demonstrations	S3	50 action steps (mean); 7.5 subgoals (mean)	PDDL-derived navigation and manipulation chains	Agent pose, object locations, receptacles, and object states	Task success and goal-condition success
AppWorld	App/API Workflows	750 tasks	S3	42.5 calls (mean; Test-N) / 46.8 (mean; Test-C)	Dependent API calls across multiple apps	Application databases, stateful execution-shell variables/results, and simulated date/time	State-based unit tests, including collateral-damage checks
SWE-Bench Pro	Professional Software Engineering	1,865 tasks	T2/S3	502 s estimated latency; 98.8 tool calls (GPT-5.4, high reasoning)	Professional requirements couple cross-file implementation choices	Large repository, dependencies, build artifacts, and tests	Repository-level acceptance and regression tests
Terminal-Bench 2.0	Terminal / Systems	89 tasks	T4–5	Expert median bucket 1 h–1 day (74 timed tasks)	Commands, debugging, builds, and artifacts depend on prior state	Filesystem, packages, processes, artifacts, and logs	Task-specific executable tests in isolated containers
SWE-Marathon	Project-scale Software Engineering	20 tasks	T5	Expert-human estimate 40–400 h	Repeated project-scale implementation, build, and test loops	Task workspace, source/output artifacts, build/runtime state, and measured results	Tests, behavioral checks, judges, and performance gates
ALE-Bench	Algorithm Engineering	40 full / 10 lite	B4	Contest-window median 4 h; agent cap 4 h/problem	Propose $\rightarrow$ implement $\rightarrow$ run $\rightarrow$ score/error feedback $\rightarrow$ refine	Current/best code, scores, feedback, and execution logs	Continuous problem score and normalized performance
Frontier SWE	Frontier Technical Work	17 tasks	B5	20 h/task (protocol budget)	Open-ended technical requirements couple implementation with task-specific functional, research, or performance objectives	Task workspace, source/checkpoints, experimental variants, and build/benchmark results	Task-specific functional and performance checks; correctness and anti-cheat gates
PostTrain Bench	AI R&D / Post-training	28 task configurations	B5	10 h on one H100/ configuration (protocol budget)	Data preparation, training, evaluation, and tuning loops	Training data, scripts/configs, checkpoints, logs, and evaluation results	Held-out target-benchmark evaluation; LLM judge for contamination and model substitution

Typed extent levels. Prefix T denotes time-derived extent, S denotes native steps, states, turns, actions, or calls, and B denotes a protocol budget or execution window. Numeric levels remain ordinal only within the same prefix and compatible measurement provenance: time bins are below 1 min, 1–10 min, 10–60 min, 1–4 h, and above 4 h; native-count bins are < 10, 10–< 30, 30–< 100, 100–< 300, and ≥ 300. Budget codes use the bin associated with the reported budget's native unit, require that unit for interpretation, and are not task-length measurements. Thus T3, S3, and B3 are not directly comparable; a slash records two modalities for the same rollout. Each extent value retains its reported mean, median, range, bucket, estimate, or budget status. The WebArena duration is an external primary remeasurement. The SWE-Bench Pro value is OpenAI's model- and reasoning-specific rollout estimate rather than a benchmark-intrinsic task-length statistic; all other values use official direct or derived evidence.

et al., 2025). The goal is not merely to ask whether an agent solves a task, but to characterize how consistently it solves it, how much progress it makes before failure, whether its trajectory is valid, what state changes it induces, and at what computational or monetary cost.

This distinction is especially important for repeated evaluation. Let $s_{ij} \in \{0, 1\}$ denote whether rollout j succeeds on task i. When exactly k independent rollouts are executed per task, two useful empirical

quantities are:

$$\widehat{\text{pass@}k} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left[\max_{1\le j\le k} s_{ij} = 1\right], \tag{6}$$

$$\widehat{\text{pass}^k} = \frac{1}{n}\sum_{i=1}^{n} \prod_{j=1}^{k} s_{ij}. \tag{7}$$

Here, $\text{pass@}k$ measures whether at least one attempt succeeds, whereas pass^k measures whether all k repeated attempts succeed. τ-bench makes this distinction explicit in tool-agent-user interaction, showing that one-shot success can substantially overstate behavioral consistency across trials (Yao et al., 2024). Beyond pass@1 further argues that reliability should be modeled as a function of task horizon, since longer tasks expose decay, variance amplification, graceful degradation, and meltdown behavior that are invisible to aggregate pass@1 (Khanal et al., 2026).

Outcome metrics should be grounded in externally checkable task states whenever possible. Benchmarks such as AgentBench (Liu et al., 2024), WebArena (Zhou et al., 2024b), OSWorld (Xie et al., 2024b), and SWE-bench (Jimenez et al., 2024) move evaluation beyond surface-form answers by tying success to interactive environments, final goal states, operating-system actions, or executable tests. This shift is essential for agents because a fluent final response may hide invalid tool calls, skipped constraints, irreversible side effects, or accidental success. Final correctness should therefore be paired with progress metrics, trajectory metrics, execution-safety metrics, latency, and cost.

The experimental unit should be an episode. Each episode should record the task instance, initial environment state, model version, scaffold configuration, tool schema, decoding parameters, random seed, full action–observation trajectory, final state, grader output, and resource vector. For each agent–benchmark pair, evaluations should include multiple rollouts per task, stratification by domain and horizon length, confidence intervals over tasks and seeds, and paired comparisons when systems are evaluated on the same task set. This protocol prevents common confounds: a higher score may reflect a stronger model, but it may also reflect more search, longer contexts, extra retries, additional verifier calls, or a more favorable harness.

Process-level evaluation is also necessary because final outcomes often underdetermine the underlying failure mode. Agent-as-a-Judge illustrates one direction: agentic evaluators can inspect intermediate trajectories and provide feedback over the task-solving process rather than judging only the final output (Zhuge et al., 2025). Such process evaluation is useful for diagnosing tool misuse, looping behavior, invalid intermediate assumptions, and partial progress. However, judge-based process scores should complement rather than replace executable validation when deterministic graders are available.

Finally, evaluation infrastructure should make resource and implementation effects explicit. Holistic leaderboards such as HAL emphasize standardized harnesses, shared logs, large-scale rollouts, and cost-aware comparison (Kapoor et al., 2025). This direction is critical for agent research: without trajectory logs, budget reporting, and harness disclosure, it is difficult to determine whether progress comes from stronger models, better scaffolds, more computation, or evaluation artifacts. A mature metric stack should therefore make agent performance attributable, reproducible, and deployment-relevant.

3.2 Web Navigation

Web navigation and browsing require agents to work in open and changing web environments. Agents must not only click, type, and search, but also track page states, check evidence, and avoid unsafe actions across long trajectories.

Interface Execution. Interface execution focuses on whether agents can turn user instructions into correct webpage operations. The main failure mode is action grounding: the agent may understand the goal but click the wrong element, fill the wrong field, or choose an invalid action. Some benchmarks use controlled or simulated websites to test step-by-step UI operation (Shi et al., 2017; Yao et al., 2022). Others extend this setting to real websites, multi-domain tasks, dialogue-based instructions, and desktop–web

Table 3: Metric stack for reliable long-horizon agent evaluation. Outcome and progress describe what happened; repeated-run, uncertainty, and boundary metrics quantify reliability; trajectory, verification, recovery, safety, and cost metrics diagnose whether the claim is valid and deployment-relevant.

Metric layer	Typical metrics	Core question	Reporting principle
Outcome	Success rate; exact match; goal-state match; pass@1	Did the agent complete the task?	Report task-level success with clear grading rules, preferably based on executable tests or final environment states.
Repeated attempts	pass@k; best-of-k success	Can the agent succeed if given multiple tries?	Report k, sampling settings, and the selection rule; distinguish best-of-k capability from single-run deployment performance.
Reliability	pass^k; trial consistency; success variance	Does the agent succeed consistently across repeated runs?	Use multiple rollouts per task and report variance or confidence intervals, since one-shot success can hide instability (Yao et al., 2024; Khanal et al., 2026).
Uncertainty	Binomial interval; task bootstrap; paired interval	How precisely is the reported reliability known?	State the resampling unit, number of tasks and rollouts, and a 95% interval; do not treat rollout-level samples from the same task as independent tasks.
Reliable boundary	$R_{S,Q}(\mathbf{z})$; $\mathcal{H}_\alpha$; $H_\alpha(g)$; robust prefix	Under which declared task pressures does reliability remain above α?	Hold the system and protocol fixed, report the stress path or conditional slice, and use the reliable set or robust prefix when difficulty is non-monotone.
Progress	Subgoal completion; milestone accuracy; partial credit	How far does the agent get before failure?	Measure intermediate progress when final success is sparse, and distinguish early collapse from near-complete execution.
Trajectory	Action validity; tool-use correctness; loop rate; process score	Was the reasoning–action path valid and efficient?	Log full trajectories and evaluate process quality in addition to final outcomes (Zhuge et al., 2025).
Verification quality	Check coverage; false accept/reject; verification delay	Can the protocol observe success and failure correctly and promptly?	Prefer executable or state-based checks; disclose judge prompts and calibration when model-based grading is unavoidable.
Recovery	Recovery rate; rollback success; actions or time to recovery	Can the system return to a valid state after an error?	Inject or label failures, distinguish detection from successful recovery, and report collateral state changes.
Execution safety	State-diff correctness; unintended actions; rollback success	Did the agent modify the external environment correctly?	Record initial and final states, specify allowed changes, and separately evaluate irreversible or harmful side effects.
Efficiency and cost	Tokens; tool calls; latency; verifier calls; monetary cost	How much resource was consumed?	Compare agents under matched budgets and report cost-normalized success or accuracy–cost trade-offs (Kapoor et al., 2025).

interfaces, where page layouts and user goals are more diverse (Deng et al., 2023; Lù et al., 2024; Xu et al., 2024b; Wang et al., 2024b; Kapoor et al., 2024).

Trajectory Robustness. Trajectory robustness focuses on whether agents can maintain a correct path over many web interactions. Here, the key problem is not a single action, but state drift: an early mistake may move the agent to the wrong page state, and later actions may look reasonable while the whole trajectory has already gone off track. Some benchmarks build functional websites or online environments to test agents under changing states (Zhou et al., 2024b; Pan et al., 2024; Xue et al., 2025; Sun and Chen, 2025). Another line adds visual grounding, where screenshots, icons, layouts, and visual cues are needed to decide the next action (Koh et al., 2024; He et al., 2024; Zheng et al., 2024a; Thomas et al., 2025). These benchmarks show why long-horizon web tasks cannot be judged only by isolated action accuracy.

Trustworthy Browsing. Trustworthy browsing focuses on whether agents can produce reliable and

safe outcomes in open web environments. The main failure mode shifts from wrong actions to wrong evidence, unsafe behavior, or misleading success. Some benchmarks require agents to search across webpages, compare sources, check claims, and generate citation-supported answers or reports (Nakano et al., 2022; Yoran et al., 2024; Wei et al., 2025; Zhou et al., 2025a; Gou et al., 2025; Du et al., 2025). Others add safety, privacy, and security requirements, where agents must avoid adversarial webpages, privacy leakage, unsafe operations, fabricated evidence, and shortcut behavior (Anupam et al., 2025; Levy et al., 2024; Tur et al., 2025; Ying et al., 2026; Ramesh et al., 2026; Li et al., 2026b).

Overall, web navigation benchmarks shift the focus from task completion to trajectory reliability. The central challenge is whether agents can maintain coherent, verifiable, and safe behavior across long interactions in open web environments.

3.3 Computer, Mobile, and Embodied Interaction

Computer-use, mobile, and embodied benchmarks become long-horizon when agents must keep what they see, what they do, and what the environment has become aligned across many dependent actions. The key failure mode is not only a wrong final answer, but gradual observation–action–state drift: a stale screen interpretation, missed object state, wrong tap, or failed manipulation can redirect the future trajectory. This distinguishes the setting from browser-only navigation, software-repository repair, and API/workplace workflows (Xie et al., 2024b; Rawles et al., 2025; Shridhar et al., 2020).

Desktop computer use. Desktop benchmarks shift evaluation from page-level navigation to full operating-system state. OSWorld serves as the anchor benchmark, with agents operating in real computer environments and interpreting screenshots, applications, windows, files, settings, and cross-application effects (Xie et al., 2024b). Windows Agent Arena adds Windows-specific realism and scalable multimodal evaluation, while OmniACT broadens visual-action grounding across heterogeneous desktop and web interfaces (Bonatti et al., 2025; Kapoor et al., 2024). Their common challenge is tracking visual feedback, file state, and application state as they evolve within a single desktop trajectory.

Mobile interaction. Mobile benchmarks add compact screens, app-specific navigation, dynamic task parameters, and device variation. AndroidWorld evaluates agents in a dynamic Android environment, where success requires maintaining app state and acting through real mobile interfaces rather than predicting a static answer (Rawles et al., 2025). B-MoCA adds the robustness angle by testing mobile device-control agents across diverse configurations, making layout, language, and settings variation part of the challenge (Lee et al., 2024). Longer trajectories require repeated UI interpretation under shifting device state, because each tap may expose a new page or alter task-relevant settings.

Embodied worlds. Embodied benchmarks make state change physical or simulated, so moving the wrong object, missing a precondition, or failing to recover can alter the future environment. ALFRED pairs household instructions with egocentric visual observations and low-level actions, requiring agents to connect language, perception, and action across everyday tasks (Shridhar et al., 2020). BEHAVIOR-1K expands scale and realism through 1,000 simulated everyday activities, while RoboCerebra moves toward long-horizon robotic manipulation with multi-step reasoning, subtask execution, and dynamic scenes (Li et al., 2023; Han et al., 2026). Planning depth matters, but long-horizon grounding also requires sustained control over a changing world, because early mistakes may remove preconditions for later actions.

Hybrid interfaces and boundaries. Hybrid benchmarks show that computer-use trajectories increasingly cross interface boundaries. WeaveBench combines GUI, CLI, code, and file-system operations, so evaluation must judge evidence across screenshots, commands, files, and intermediate artifacts rather than endpoint state alone (Li et al., 2026b). Its mixed interface places it near software and terminal evaluation, but the central difficulty remains cross-interface grounding rather than executable software-engineering correctness. HeroBench illustrates the boundary from the planning side. Virtual worlds may contain long action sequences and structured reasoning, but when resource dependencies, numerical feasibility, and strategic planning dominate, they fit better with constraint-heavy planning than computer-use grounding (Anokhin et al., 2025). These tasks therefore test whether agents can track visual evidence, executed actions, and environment state as interfaces change over time.

3.4 Software Engineering and Terminal Tasks

Software-engineering and terminal benchmarks become long-horizon when coding or command-line work is no longer a one-shot generation problem but a stateful process in an executable environment. Agents must maintain a model of workspace state, coordinate edits across files, use terminal, build, test, and CI feedback, and avoid regressions. Executable work environments expose these demands directly, as repository-level issue repair and terminal-based task environments make state tracking, feedback use, and validation part of the benchmarked trajectory (Jimenez et al., 2024; Merrill et al., 2026).

Scaling repository repair. SWE-bench established repository-level issue repair as the canonical protocol, and its Verified subset further tightened task quality and evaluation reliability (Jimenez et al., 2024; OpenAI, 2024). Subsequent variants show that this protocol is still evolving. SWE-Cycle makes the hidden lifecycle explicit, including environment reconstruction and verification-test generation (Guan et al., 2026). SWE-bench-Live, SWE-MERA, and SWE-rebench V2 address staleness, contamination, language coverage, and data-scale pressures (Zhang et al., 2026a; Pavel et al., 2025; Badertdinov et al., 2026). The same line then escalates through SWE-Bench Pro's harder professional tasks, DeepSWE's original long-horizon tasks with program-based verifiers, and SWE-Marathon's ultra-long executable environments with multi-layer verification (Deng et al., 2025; Huang et al., 2026; Desai et al., 2026).

Beyond issue repair, evaluation includes tasks that require agents to understand, extend, and construct software systems. FeatureBench targets complex feature implementation, while SWE-Explore and ReCUBE isolate two pre-editing bottlenecks: localizing the relevant code and using repository context correctly (Zhou et al., 2026b; Zhang et al., 2026c; Hong et al., 2026). SWE Atlas adds codebase question answering, test writing, and refactoring. ProgramBench, NL2Repo-Bench, and RepoGenesis evaluate whole-program or whole-repository construction (Raghavendra et al., 2026; Yang et al., 2026b; Ding et al., 2026; Peng et al., 2026).

Maintained correctness over time. Long-horizon software work also tests whether changes remain correct as repositories evolve. A patch can pass immediate tests yet fail under future requirements, version transitions, CI changes, or test-suite evolution. SWE-EVO therefore evaluates software evolution from release notes and mature project histories, requiring coordinated multi-file modifications while preserving existing behavior (Le et al., 2026). SWE-CI shifts the emphasis to maintainability under CI-style repository evolution, while SWE-Milestone evaluates dependency-constrained milestone streams in a persistent codebase, making cross-task error propagation and regression accumulation directly measurable (Chen et al., 2026; Deng et al., 2026). RoadmapBench and SWE-Chain formulate long-horizon upgrades as version or release chains in which earlier changes become the starting state for later ones (Xu et al., 2026e; Lam et al., 2026). ChainSWE extends this setting to sequential, dependent bug fixes in a shared repository, so later tasks inherit the agent's earlier changes (Jin et al., 2026a). TEBench complements this line by treating the maintained artifact as not only production code, but also the tests that specify and protect behavior (Shang et al., 2026).

Terminal and CI as validation. Repository success depends on the machinery that proves code works. Terminal-Bench evaluates agents in real command-line environments, and LongCLI-Bench narrows this setting to programming tasks performed through the CLI (Merrill et al., 2026; Feng et al., 2026b). BuildBench, DI-BENCH, and CI-Repair-Bench show that executable validation often begins with environment management rather than code generation, requiring agents to infer dependencies, diagnose build logs, repair configuration or workflow failures, and rerun the system under build, dependency, and CI constraints (Zhang et al., 2025b,a; Muna et al., 2026). CLI-Tool-Bench and SWE-InfraBench extend CLI-based evaluation to command-line tool generation and infrastructure-as-code edits (Hu et al., 2026; Tarasova et al., 2026). These tasks connect code changes to executable evidence, making setup, failure diagnosis, and validation part of the evaluated trajectory.

Frontier engineering and AI-R&D automation. Recent repository and terminal benchmarks extend the evaluation target from issue repair and code maintenance to longer engineering tasks. FrontierSWE emphasizes multi-hour tasks in performance engineering, computational science, and machine-learning systems (Chu et al., 2026). PostTrainBench evaluates agents that combine code, search, data curation, training, and evaluation loops to improve a base model under a fixed compute budget (Rank et al., 2026).

These settings still rely on executable artifacts, command-line feedback, and verification, but the evaluated outcomes are open-ended engineering results and AI-R&D pipelines rather than ordinary patch production.

The resulting benchmark family treats long-horizon SWE evaluation as a problem of sustained control over changing repositories, executable environments, and imperfect verification processes, not merely as harder coding.

3.5 Tool, API, and Workplace Automation

Tool- and application-use benchmarks test whether agents can remain reliable across many dependent actions. The core difficulty is not merely selecting the right tool, but preserving state, carrying information across steps, and avoiding unintended changes as the workflow unfolds.

From function calls to stateful interaction. Early benchmarks focused on tool selection and argument generation. ToolBench, introduced with ToolLLM, scales evaluation to 16,464 RapidAPI endpoints, while BFCL measures function-call correctness through structured matching and execution-based checks (Qin et al., 2024; Patil et al., 2025). These benchmarks establish basic tool-use competence, but short-horizon call accuracy does not directly predict performance on longer workflows, where early mistakes propagate to later steps.

AppWorld shifts the evaluation target from individual calls to persistent application state. Agents interact with 457 APIs across nine applications, and state-based unit tests verify both task completion and *collateral damage* to unrelated data (Trivedi et al., 2024b). This captures a central requirement of long-horizon tool use: reaching the intended state without corrupting the surrounding environment.

From single applications to extended workflows. WorkArena evaluates enterprise tasks on the ServiceNow platform, introducing realistic interfaces and structured business processes (Drouin et al., 2024). OfficeBench broadens the action space to workflows spanning Word, Excel, PDF, email, calendar, and shell tools (Wang et al., 2024e). Such tasks require agents to transfer information across applications and maintain consistency over multiple state transitions.

OdysseyBench adds long-range conversational dependencies: agents must recover relevant information from extended interaction histories and use it to complete multi-application tasks (Wang et al., 2025c). Its mixture of 300 transformed workflows and 302 automatically generated tasks also illustrates a recurring realism–scale trade-off: synthetic construction increases coverage, but requires careful validation of task quality and representativeness.

Toward integrated workplaces. TheAgentCompany combines browsing, coding, enterprise services, and communication with simulated coworkers in a single organizational environment (Xu et al., 2026b). Its checkpoint-based evaluation distinguishes partial progress from full completion; even the strongest reported agents complete only about 30% of tasks end to end. This gap suggests that current systems can often solve isolated subtasks but struggle to maintain coherence across an entire workflow.

Overall, these benchmarks expose three related dimensions of difficulty: *compositional depth*, *persistent state*, and *environmental breadth*. Evaluation has accordingly moved from call-level matching toward executable checks, state assertions, and checkpointed progress. However, current metrics remain weak at identifying process-level failures, such as stale-state reasoning, failed recovery, unnecessary side effects, and procedural violations. Developing evaluators that localize these failures without penalizing valid alternative trajectories remains an important open challenge.

3.6 Planning and Scientific Work

Where the web, computer-use, and software settings mostly stress getting each interface action right, the benchmarks in this group stress keeping a plan globally consistent as constraints, dependencies, and intermediate results accumulate. A step that looks locally reasonable can still leave the final plan infeasible. Many of these benchmarks also involve heavy tool use and database queries, but we group them by their dominant source of difficulty, global constraint satisfaction.

Real-world and natural-language planning. TravelPlanner (Xie et al., 2024a) asks an agent to assemble a multi-day itinerary from a database of roughly four million records, satisfying explicit budget and preference constraints together with implicit commonsense ones; across 1,225 queries even GPT-4

returns a fully valid plan in just 0.6% of cases, and ReAct- or Reflexion-style strategies break down once several constraints interact. NATURAL PLAN (Zheng et al., 2024b) asks whether the bottleneck is tool use or planning itself: it feeds tool outputs such as Google Flights, Maps, and Calendar directly into the context, removing the tool-use step, yet accuracy still collapses with scale, falling below 5% once ten cities are involved, and self-correction does not help. DeepPlanning (Zhang et al., 2026d) keeps a similar scenario but tightens the evaluation, grading minute-level travel and shopping plans with rule-based checkers in offline sandboxes and separating proactive information gathering from local and global constraint reasoning. Its error analysis is especially detailed, showing that frontier agents skip required tool calls on long horizons and rarely check global consistency or backtrack.

Difficulty built into a simulated world. Because the difficulty of the real-world suites is inherited from the domain, no single source of hardness can be isolated or varied on its own: TravelPlanner can only separate planning from tool use by adding a second, sole-planning mode, and NATURAL PLAN has to remove tools from the task altogether. The benchmarks in this group instead construct the difficulty themselves, introducing a chosen kind of structure that can be controlled directly. MinePlanner (Hill et al., 2023) formalizes 45 Minecraft building and navigation tasks in PDDL, the standard language for classical planning, where the hardest instances contain thousands of objects, most of them irrelevant to the goal. Its difficulty is not specific to LLMs: even dedicated classical planners run out of memory or hit the time limit on these instances before returning a plan. Robotouille (Gonzalez-Pumariega et al., 2025) adds timing constraints. Its cooking tasks run with real delays and interruptions, so an agent must interleave subtasks instead of running them in sequence, and the cost of failing to do so is steep, with ReAct on GPT-4o falling from 47% on synchronous tasks to 11% on asynchronous ones.

From feasibility to solution quality. In every benchmark so far, the verdict is binary: a plan is feasible or it is not. This group turns to algorithmic engineering, where the question is how good a solution is. ALE-Bench (Imajuku et al., 2026) grades each attempt with a score the agent improves over a long session, drawing its problems from AtCoder Heuristic Contests. Frontier models can match strong human contestants on a single problem, but they fall behind on the two things the format rewards most: consistency across problems and steady improvement over time. Across these benchmarks, a single success rate reveals little about why agents fail. The same failure modes appear across domains: each constraint is satisfied on its own but the plan violates them together, intermediate results get lost, a plan that looks plausible fails the checker, and an early mistake spreads until it can no longer be fixed. The benchmarks that catch these failures most reliably are the ones with automatic verifiers, such as TravelPlanner's constraint checks, DeepPlanning's sandboxes, and ALE-Bench's scorer. The model-level methods and harness-level mechanisms we review next can be judged against the same kind of automatic check.

4 Model Design: Reducing Local Decision Error

Long-horizon failure is often a compounding rather than a single catastrophic process. Under the deliberately simplified assumption of independent and identically distributed step outcomes, a 95% per-step success rate over 100 actions gives $0.95^{100} < 1\%$ end-to-end success. This calculation is illustrative, not a trajectory model: real agent errors are state-dependent, temporally correlated, non-stationary, and may either cascade or be contained by feedback and recovery. Section 5 examines how external execution infrastructure can compensate for model errors; this section asks what capabilities, internalized through inference-time computation or parameter updates, reduce local transition error in the first place. We organize these model-level determinants along three axes: deliberate reasoning and search at inference time (§4.1), supervised internalization of tool use (§4.2), and reinforcement learning over full interaction trajectories (§4.3). Figure 4 maps these axes and their representative mechanisms, while Table 4 summarizes their long-horizon roles and limitations.

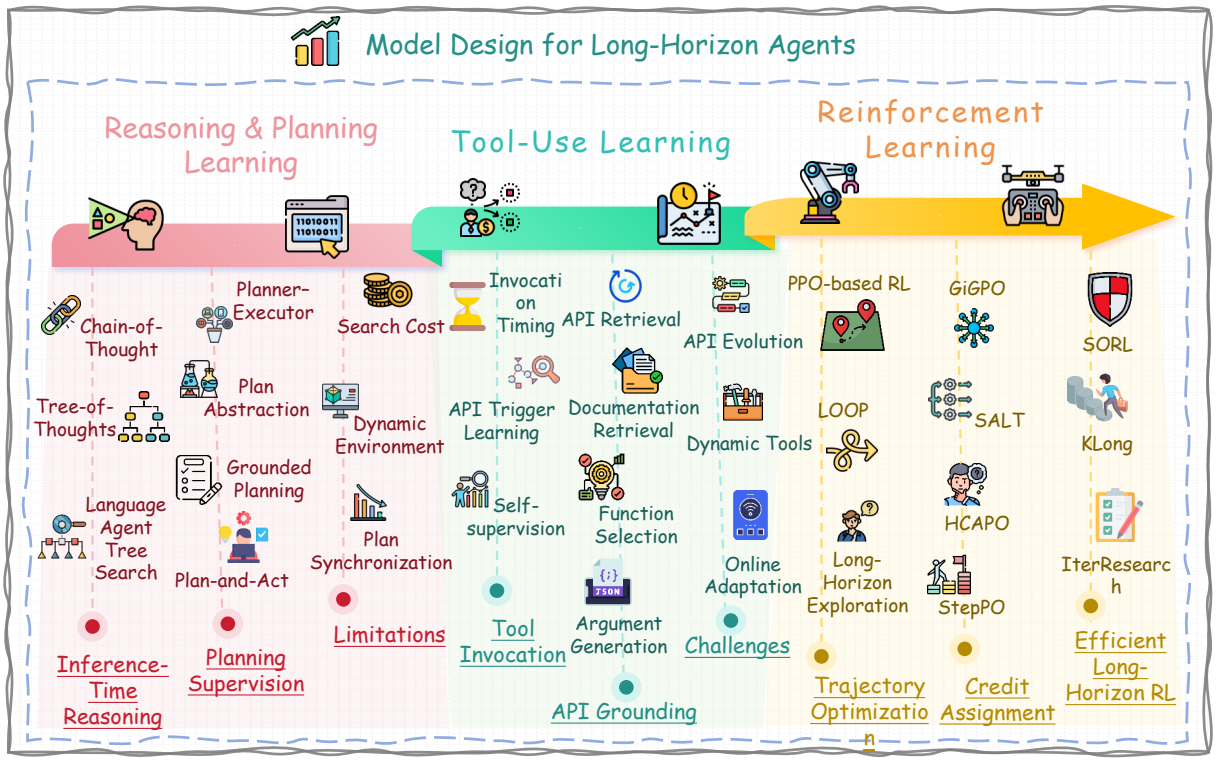

Figure 4: Model-design mechanisms for reducing local decision error in long-horizon agents. Reasoning and planning interventions progress from inference-time reasoning and search to grounded planning supervision; tool-use learning covers invocation timing, API grounding, and adaptation to changing tools; trajectory-level reinforcement learning addresses policy optimization, credit assignment, stability, and horizon scaling. The diagram is a conceptual taxonomy rather than a quantitative comparison.

4.1 Reasoning and Planning Supervision

The earliest model-level interventions for reducing planning errors operate at inference time, requiring no parameter changes. **Chain-of-Thought (CoT) prompting** (Wei et al., 2022) elicits explicit intermediate reasoning steps before an agent commits to an action, reducing the frequency of locally incoherent decisions on arithmetic, commonsense, and symbolic tasks. While CoT externalizes single-path reasoning, it does not recover from dead ends. Tree of Thoughts (ToT) (Yao et al., 2023a) organizes intermediate states into a search tree, enabling backtracking and deliberate evaluation of alternative reasoning branches via breadth-first or depth-first search. ToT demonstrates that allocating more inference-time compute to structured search can substitute for weaker per-step reasoning.

Both CoT and ToT operate over reasoning traces disconnected from external environments. Language Agent Tree Search (LATS) (Zhou et al., 2023) bridges this gap by integrating Monte Carlo Tree Search (MCTS) with the ReAct (Yao et al., 2023c) loop, interleaving reasoning, grounded action execution, and language-based reflection within a unified tree search framework. LATS introduces environment feedback as a signal for tree-node evaluation, enabling recovery from execution errors rather than just reasoning errors.

Inference-time search methods are powerful, but their main model-level value lies in the traces they can provide for training. Plan-and-Act (Erdogan et al., 2025) further casts planning as a midtraining data problem. Rather than only separating a Planner from an Executor, it constructs grounded planning supervision from successful environment trajectories: a teacher model abstracts high-level plan steps from executed action traces and aligns them with the low-level actions that realize each step. This query–plan and plan–action data is then used to train models to produce executable decompositions instead of ungrounded plans. Empirically, high-quality dynamic planning substantially improves even a base Executor, raising WebArena-Lite success from 9.85% to 44.24%, and the full system reaches 57.58%.

The progression from CoT to LATS to Plan-and-Act reveals a consistent pattern: each method addresses

the limitations of the previous one by introducing additional structure, including tree search, role specialization, and training. However, each also introduces new costs: compute overhead for search, sensitivity to evaluator quality, and the challenge of keeping plans synchronized with dynamic environments.

4.2 Tool-Use Learning

In grounded agent settings, planning errors and tool-use errors are distinct failure modes. An agent may reason correctly about what to do yet still select the wrong API, generate invalid arguments, or misinterpret a tool's return value. The methods in §4.1 generally assume tool invocation is straightforward; this subsection asks what it takes to internalize reliable tool use through training.

Toolformer (Schick et al., 2023) pioneers self-supervised tool-use acquisition: the model generates its own training data by inserting candidate API calls into existing text, filtering those that reduce perplexity on subsequent tokens, and fine-tuning on the resulting corpus. This approach allows a model to learn *when* to call a tool without human annotation of tool-use opportunities. However, Toolformer's self-supervised signal is tied to next-token prediction and does not optimize for task-level outcomes across multi-step interactions.

As APIs proliferate, retrieval becomes essential. Gorilla (Patil et al., 2024) fine-tunes a model specifically for API invocation accuracy over a catalog of 16,000+ real-world APIs. By combining retrieval-augmented generation with instruction fine-tuning, Gorilla substantially reduces hallucination of invalid function signatures and achieves stronger performance than GPT-4 on API invocation benchmarks when documentation is retrieved rather than memorized.

Together, Toolformer and Gorilla establish two complementary principles: learning *when* to invoke tools as a temporal trigger and *which* tool to invoke accurately as API grounding. Both remain limited by their static training distributions: neither adapts dynamically when APIs change or deprecate. This limitation motivates environment-embedded RL approaches in §4.3.

4.3 Reinforcement Learning over Long Trajectories

Supervised learning of reasoning and tool use reduces local step errors, but it does not directly optimize for long-horizon task completion. Distributional gaps between training trajectories and real deployment, compounding errors across steps, and the absence of dense per-step supervision collectively motivate reinforcement learning over full agent trajectories. Agentic RL presents four distinct challenges: sparse and delayed outcome rewards, credit assignment, training stability under off-policy updates, and context scaling when trajectories exceed context-window limits.

Trajectory-level training. LOOP (Chen et al., 2025b) establishes that RL training in stateful, multi-domain environments is feasible for LLM agents. Training interactive digital agents (IDAs) to use APIs of persistent application environments as a partially observable MDP, LOOP derives a memory-efficient PPO variant that maintains a single copy of the LLM without a value network. A 32B agent trained with LOOP on AppWorld surpasses the much larger OpenAI o1 by 9 percentage points, and analysis reveals that RL causes the agent to learn substantively useful behaviors: consulting API documentation, avoiding unwarranted assumptions, and recovering from execution failures.

Step-level credit assignment. A central difficulty in applying group-based RL, such as GRPO, to agent tasks is that sparse terminal rewards are assigned uniformly across all trajectory steps. Beneficial and detrimental actions receive identical reward signals when they co-occur in the same trajectory, inducing gradient conflicts during policy optimization.

GiGPO (Feng et al., 2026a) addresses this with a two-level group structure: at the episode level, macro advantages are computed from groups of complete trajectories sharing the same task; at the step level, an anchor-state grouping mechanism identifies repeated environment states across different rollouts and groups actions arising from the same state, enabling step-level micro-advantage estimation. GiGPO achieves greater than 12% improvement over GRPO on ALFWorld and greater than 9% on WebShop while maintaining identical GPU memory overhead and rollout cost.

SALT (Li et al., 2026a) constructs a trajectory graph from multiple rollouts of the same prompt, using cross-trajectory structural information to quantify the relative quality of each step and assign advantages

accordingly. SALT is designed as a plug-and-play enhancement to existing group-based RL algorithms: it integrates with GRPO and RLOO without modifying the rollout procedure, introducing negligible computational overhead. Experiments on WebShop, ALFWorld, and AppWorld confirm consistent performance gains across model sizes.

HCAPO (Tan et al., 2026) takes a generative approach to credit: it uses the LLM itself as a post-hoc critic, injecting the known successful outcome into the prompt to simulate a hindsight distribution and refine step-level Q-value estimates through generative verification. A multi-scale advantage mechanism additionally corrects misaligned value baselines at critical decision states. StepPO (Wang et al., 2026b) approaches the same granularity mismatch from a different angle: rather than refining rewards, it reformulates the agentic MDP at the step level, treating interaction steps rather than tokens as the basic trajectory units for both modeling and policy optimization.

Stability and context scaling. Off-policy training exacerbates instability in long-horizon settings: token-level importance-sampling weights diverge as policy distributions drift across gradient updates, producing high-variance gradients and performance collapse. SORL (Li et al., 2025a) identifies two compounding sources of instability—token-level granularity mismatch and off-policy variance—and addresses them with turn-level importance sampling and clipping-triggered normalization. By computing importance weights at the turn boundary rather than the token level, SORL aligns credit correction with the natural interaction granularity of multi-turn agents, yielding consistently stable training curves and preventing the collapse observed in standard PPO and GRPO.

For tasks measured in hours of human effort, trajectories can span hundreds of turns and hundreds of thousands of tokens, far beyond standard context windows. KLong (Liu et al., 2026b) addresses this with a two-stage training recipe. First, trajectory-splitting SFT divides expert trajectories, distilled from Claude Sonnet with thinking, into overlapping sub-trajectories that preserve early context, progressively truncate later content, and maintain continuity across splits. Second, progressive RL schedules training across multiple stages with incrementally extended timeouts ($2 \rightarrow 4 \rightarrow 6$ hours), allowing the agent to gradually adapt to tasks with increasingly delayed reward structures. KLong at 106B parameters outperforms Kimi K2 Thinking (1T parameters) by 11.28% on PaperBench, demonstrating strong generalization to SWE-bench Verified and MLE-bench.

A complementary strategy addresses context growth at inference time. Mono-contextual agents append all retrieved information and reasoning steps to a single expanding context, leading to what IterResearch (Chen et al., 2025a) terms *context suffocation*: as context grows, earlier relevant information becomes harder to attend to, and noise accumulates. IterResearch reformulates long-horizon research as a Markov Decision Process in which the agent maintains an evolving workspace report rather than a raw context log, periodically synthesizing and reconstructing state. This Markovian reconstruction decouples exploration depth from context length, enabling scaling to 2048+ tool calls while holding context to approximately 40K tokens. Performance improves dramatically with interaction count, from 3.5% to 42.5%, a scaling law not observed in mono-contextual baselines. IterResearch further develops Efficiency-Aware Policy Optimization (EAPO), which applies geometric reward discounting to incentivize efficient exploration and uses adaptive downsampling for stable distributed training.

Having examined the three mechanism families separately, Table 4 consolidates their representative implementations, long-horizon roles, and primary limitations. Unlike the conceptual progression in Figure 4, the table is intended as a method-level reference.

Table 4: Representative model-level mechanisms for reducing per-step error. The regimes move from inference-time structure to tool-use internalization and long-horizon policy optimization.

Regime	Representative methods	Long-horizon role	Primary limitation
Inference-time reasoning	CoT, ToT, LATS (Wei et al., 2022; Yao et al., 2023a; Zhou et al., 2023)	Adds intermediate reasoning, backtracking, and environment-grounded tree search before committing to actions.	High inference cost; dependent on evaluator quality.
Explicit planning	Plan-and-Act (Erdogan et al., 2025)	Separates high-level planning from environment-specific execution, reducing plan-generation bottlenecks.	Plans must remain synchronized with dynamic environments.
Tool-use learning	Toolformer, Gorilla (Schick et al., 2023; Patil et al., 2024)	Internalizes when to call tools and how to ground API invocation in documentation.	Static API distributions; weak task-level optimization.
Trajectory-level RL	LOOP (Chen et al., 2025b)	Optimizes agents directly in stateful interactive environments.	Coarse trajectory-level rewards and environment dependence.
Step-level credit	GiGPO, SALT, HCAPO, StepPO (Feng et al., 2026a; Li et al., 2026a; Tan et al., 2026; Wang et al., 2026b)	Assigns advantages to steps or states rather than uniformly to whole trajectories.	Often evaluated on moderate horizons; may require repeated states or multiple rollouts.
Stability and horizon scaling	SORL, KLong, IterResearch (Li et al., 2025a; Liu et al., 2026b; Chen et al., 2025a)	Stabilizes off-policy updates and scales training or inference to longer trajectories.	Expensive curricula, domain specificity, and difficult model-harness attribution.

5 Harness Design: Sustaining Reliable Execution

Harness-level determinants concern the execution infrastructure that turns a language model into a stateful, tool-using, and evaluable agent. Model-level methods lower per-step error, but long-horizon execution fails through *accumulation*: even a high per-step success rate yields end-to-end reliability that decays as a task lengthens—empirically close to a constant per-step hazard at the aggregate level, giving each agent a characteristic success "half-life" (Ord, 2025). This aggregate regularity is compatible with the state-dependent, correlated errors of §4: a constant hazard is what one would observe if failures were approximately independent, and we read it as indirect evidence that a well-designed harness converts correlated failures into independently recoverable events. The *reliable* horizon is thus governed less by single-step competence than by whether the surrounding system contains and recovers from the errors that inevitably occur. Recent work formalizes this system as a "harness"—an externalization layer of execution loop, context manager, state store, and verification interface (Meng et al., 2026; Zhou et al., 2026a)—and suggests it can push reliability far beyond a bare model: decomposition with per-step error correction has executed a highly decomposable task of over a million steps with zero final errors (Meyerson et al., 2025). We organize the harness along three long-horizon-specific axes: grounding the execution loop in feedback, so per-step errors are caught and corrected before they cascade (§5.1); managing and persisting goal-relevant state—from the live context window to cross-episode memory and reusable skills—as the trajectory outgrows the window and spans episodes (§5.2); and sustaining *verified* autonomy by shifting outcome verification from the human to the agent (§5.3).

5.1 Execution Control and Environmental Feedback

The foundational harness pattern is an execution loop that constructs the current state, elicits reasoning or an action, executes it, observes feedback, and conditions the next step on the result—turning one-shot generation into situated, feedback-driven execution. Within a *context-contained* episode, where the task state, recent observations, and stopping condition still fit the model's context, the loop improves reliability at minimum by making intermediate state visible and turning failures into signals. ReAct interleaves reasoning traces with environment-facing actions (Yao et al., 2023c), and closed-loop natural-language

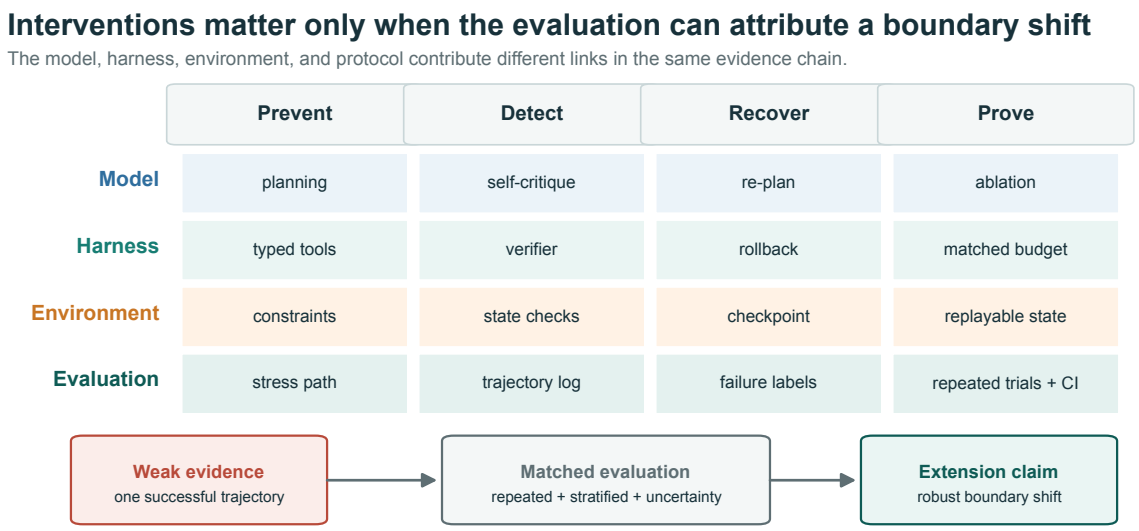

Figure 5: Intervention-to-evidence chain for reliable-horizon claims. Model and harness choices can prevent, detect, or recover from errors, while replayable environments and matched evaluation protocols make those effects attributable. A single successful trajectory is weak evidence; a credible extension claim requires repeated, stratified, matched evaluation with uncertainty and an outward shift of the reliable boundary. The diagram is schematic.

feedback lifts long-horizon embodied instruction-following without additional training (Huang et al., 2023). Feedback-driven refinement makes correction explicit: Reflexion converts outcome signals into verbal reflections (Shinn et al., 2023), Self-Refine iterates generate–feedback–refine (Madaan et al., 2023), Chain-of-Verification and Self-Debug check drafts against verification questions or execution results (Dhuliawala et al., 2024; Chen et al., 2024), and CRITIC grounds critique in external tools (Gou et al., 2024). Adaptive replanning and plan–execute decoupling keep the loop valid and efficient as horizons grow (Sun et al., 2023; Xu et al., 2023), while search-structured variants explore and backtrack over alternative paths when local decisions are ambiguous (Yao et al., 2023b; Zhou et al., 2024a).

These methods share one assumption—that the evidence needed to evaluate and repair an action stays visible in the current state—and, for those relying on the model's own feedback, one limit: intrinsic self-correction is unreliable when the model cannot tell it has erred. A critical survey finds that self-correction from prompted-model feedback alone shows little benefit under sound evaluation settings and works mainly when *reliable external feedback* is available (Kamoi et al., 2024). The practical response is to make the corrective signal external and trainable: learned critics catch more bugs than unaided human reviewers (McAleese et al., 2024), and generative process reward models verify each intermediate step rather than only the final answer (Khalifa et al., 2026). This shift—from internal-and-late toward external-and-early correction—is the first-order lever for long horizons, and it carries directly into the verification machinery of §5.3.

5.2 Context, Memory, and Persistent State

The feedback loop of §5.1 assumes that the episode remains *context-contained*: the evidence needed to evaluate and choose the next action stays inside the context window. Long-horizon tasks break this assumption. Tool outputs, observations, code diffs, logs, and partial plans accumulate until the relevant state exceeds what the window can hold, while earlier decisions stay causally relevant long after they scroll out of view. The trajectory itself then becomes an object of harness control—selecting, compressing, externalizing, and recovering state under a finite context budget (Wang et al., 2026d). Throughout this subsection we consider in-context and externalized state; parametric memory acquired through weight updates is a model-level mechanism and belongs to §4. Table 5 summarizes the mechanism families, their persistence scope, and their binding limits.

Four mechanism families manage state within the active episode. *Observation selection and pruning*

trims high-volume tool outputs, logs, DOMs, or code before they overwhelm the next call (Wang et al., 2026e; Lindenbauer et al., 2025). *Trajectory summarization and folding* turns action–observation histories into compact continuation states (Wu et al., 2025; Sun et al., 2025). *In-context memory construction and recovery* preserves facts, constraints, and dependencies that resurface many steps later within the same run (Zhou et al., 2025b; Wu et al., 2026a). *Evidence and plan externalization* keeps inspectable artifacts, citations, tests, and evolving plans outside the prompt (Li et al., 2025b; Wu et al., 2026b). These families differ in the granularity at which they act—individual observations, whole trajectory segments, extracted facts, or external artifacts—but share a single design question: which state to keep at what cost. How state is selected, compressed, and recovered is itself a determinant of the reliable horizon: compression is inherently lossy—content discarded as irrelevant may be exactly what a later step needs (Wang et al., 2026d)—and when the harness keeps the wrong evidence, compresses away an exact constraint, or fails to restore an earlier decision, later calls can be locally reasonable yet globally invalid.

These mechanisms keep a *single* trajectory usable; memory proper persists goal-relevant state *across* episodes and sessions. Following the working/episodic/semantic/procedural taxonomy of Sumers et al. (Sumers et al., 2024), as adopted by subsequent memory surveys (Hu et al., 2025b), we group mechanisms by function into three families, merging working with semantic memory (both retain facts beyond the active window) and treating episodic storage as experiential. *Working and factual memory* retains and pages facts: Generative Agents couples a recency-weighted memory stream with periodic reflection (Park et al., 2023), MemGPT treats the context window as virtual memory with OS-style paging (Packer et al., 2023), HippoRAG indexes experience into a knowledge graph for multi-hop recall (Gutiérrez et al., 2024), and Mem0 extracts and consolidates salient facts for production use, with self-reported gains in accuracy, latency, and token cost over a proprietary baseline (Chhikara et al., 2025). *Experiential memory* stores reusable episodes: ExpeL distills natural-language insights from past trajectories without weight updates (Zhao et al., 2024), A-MEM maintains an evolving Zettelkasten of dynamically linked notes (Xu et al., 2026d), and ACE curates accumulated experience as an evolving playbook (Zhang et al., 2026b). *Procedural memory and skill libraries* store *how* to act: Agent Workflow Memory induces reusable workflows from prior trajectories (Wang et al., 2025e), Memp studies the build–retrieve–update lifecycle of procedural memory (Fang et al., 2026), and Voyager and JARVIS-1 accumulate executable, verified skills as composable building blocks for open-ended tasks (Wang et al., 2024a, 2025f).

Across all three functions, externalized state survives context limits and amortizes re-derivation over long or repeated tasks. Its binding constraint is that memory can mislead as easily as it helps: no single memory architecture dominates across workloads—flat stores, graphs, paged hierarchies, and skill libraries each win where their structure matches the task's bottleneck (Zhou et al., 2026c)—and a memory whose structure mismatches the task degrades in practice into stale or noisy state, itself a primary long-horizon failure mode that §6.2 examines as memory failure and goal drift.

Table 5: State management and memory mechanisms for long-horizon execution. Rows above the rule manage the active trajectory under a finite context budget (within-episode); rows below persist state across episodes and tasks, following the taxonomy of Sumers et al. (Sumers et al., 2024) with working and semantic memory merged as factual, and episodic memory treated as experiential.

Mechanism family	Scope	Structure and representative works	Binding limit
Observation selection & pruning	within-episode	observation-level: SWE-Pruner, Squeez, LongCodeZip, ACON, LCoW, observation masking (Wang et al., 2026e; Kovács, 2026; Shi et al., 2025; Kang et al., 2025; Lee et al., 2025; Lindenbauer et al., 2025)	relevance is task-conditioned; pruned content may be needed later
Trajectory summarization & folding	within-episode	trajectory-level: ReSum, AgentFold, Context-Folding, Trajectory Reduction (Wu et al., 2025; Ye et al., 2025; Sun et al., 2025; Xiao et al., 2026)	compression is lossy; exact constraints may be discarded
In-context memory construction & recovery	within-episode	fact/constraint-level: MEM1, ContextWeaver, proactive extraction (Zhou et al., 2025b; Wu et al., 2026a; Yang et al., 2026a)	must anticipate which state will resurface
Evidence & plan externalization	within-episode	artifact-level: WebWeaver, Step-DeepResearch, ContextBudget (Li et al., 2025b; Hu et al., 2025a; Wu et al., 2026b)	externalized state must be re-grounded before use
Working / factual memory	cross-episode	memory stream (Generative Agents), paged hierarchy (MemGPT), knowledge graph (HippoRAG), consolidated facts (Mem0) (Park et al., 2023; Packer et al., 2023; Gutiérrez et al., 2024; Chhikara et al., 2025)	staleness and retrieval precision
Experiential memory	cross-episode	distilled insights (ExpeL), linked notes (A-MEM), evolving playbook (ACE) (Zhao et al., 2024; Xu et al., 2026d; Zhang et al., 2026b)	insights may not transfer across task distributions
Procedural memory & skills	cross-task	induced workflows (AWM), lifecycle management (Memp), verified skill libraries (Voyager, JARVIS-1) (Wang et al., 2025e; Fang et al., 2026; Wang et al., 2024a, 2025f)	skills require validity maintenance as environments change

5.3 Verification, Recovery, and Autonomous Loops

Feedback and state keep a loop coherent; running it *long and unattended* additionally requires deciding, without a human, when output is good enough and when to stop. We term this *looping engineering*, following the harness perspective of Meng et al. (Meng et al., 2026): the agent finds work, executes, verifies its own outcome, and decides whether to continue, retry, escalate, or halt—moving the human from "in the loop" (reviewing each result) to "on the loop" (maintaining the harness). Verification is the crux, and the hard part. As models strengthen, reliable verification becomes harder than generation and no fixed reward survives a continually improving policy, so the verifier must co-evolve with the agent (Wang et al., 2026a); moreover, fixed test-based gates under-specify intent on long-horizon, systems-level work, leaving room for reward hacking that grows with the gap between task difficulty and model capability (Zhao et al., 2026a). A loop also cannot safely be its own judge—letting the generating model gate itself tends to ratify its own mistakes (Kamoi et al., 2024; Huang et al., 2024)—so verification is best delegated to an independent checker, where independence may be of weights (a separate model) or, more weakly, of context (a fresh view over the same model).

Agent-side verification and control. Several mechanisms supply this. Agent-as-a-Judge evaluates an agent's full trajectory with another agent rather than scoring only the endpoint (Zhuge et al., 2025); self-verifying coders interleave generation with self-generated tests (Jin et al., 2026b); and process reward models score intermediate steps so the loop can prune or accept them at inference time (Choudhury, 2025). Knowing when to *stop* matters as much as what to do next: explicit early-exit policies break repetitive loops and hand off rather than churn (Lu et al., 2025), adversarial analysis shows that corrupting progress signals can trap agents in unbounded loops (Xu et al., 2026c), and scheduler-style control flow replaces the implicit loop with bounded, inspectable execution (Wei, 2026; Sypherd and Belle, 2024).

When confidence is low the agent should escalate rather than proceed; studies of clarification timing find that frontier agents rarely ask in the right window (Gulati et al., 2026), and human-on-the-loop systems operationalize where to place such gates (Mozannar et al., 2025).

Evidence, limits, and failure. These ingredients let agents sustain coherent work over long durations: a self-reflective coding agent that verifies its own progress from an independent context has run unattended for tens of hours (Orimo et al., 2025). The same evidence cautions against equating autonomy with capability. A controlled scaling study finds that adding agents or coordination yields diminishing returns once a single-agent baseline is already strong, and that architectures lacking centralized verification amplify rather than contain errors (Kim et al., 2026); correspondingly, leaner designs—a fixed pipeline, or a production harness that is mostly a simple while-loop wrapped in operational infrastructure—often match elaborate ones at lower cost (Xia et al., 2025; Liu et al., 2026a; Wang et al., 2025b). Where loops still fail, empirical taxonomies trace the cause to non-termination, looping, and missing verification (Cemri et al., 2025; Zhu et al., 2025), motivating intervention-driven recovery that localizes the implicated step and reruns to confirm a repair (Ma et al., 2025). Table 6 organizes these mechanisms by the decision each supplies and the limit that constrains it.

Table 6: Looping engineering: the decisions an unattended loop must make, the mechanisms that supply each decision, and the binding limit that constrains it. Evidence on sustained autonomy and its failure modes (PARC; scaling caveats; failure taxonomies) is discussed in the text.

Loop decision	Representative mechanisms	Binding limit
Verify the outcome	trajectory-level judging (Agent-as-a-Judge), self-generated test verification (ReVeal), step-level process reward models (Zhuge et al., 2025; Jin et al., 2026b; Choudhury, 2025)	verification, not generation, is the binding bottleneck as capability grows; under-specified test gates admit reward hacking on long-horizon systems-level tasks (Wang et al., 2026a; Zhao et al., 2026a)
Stop or continue	early-exit policies, scheduler-style bounded control flow (Lu et al., 2025; Wei, 2026; Sypherd and Belle, 2024)	progress signals can be poisoned into unbounded loops (Xu et al., 2026c)
Escalate to a human	approval gates and human-on-the-loop interfaces (Magentic-UI) (Mozannar et al., 2025)	frontier agents rarely ask in the right window (Gulati et al., 2026)
Localize and repair	intervention-driven debugging with rerun confirmation (Dover) (Ma et al., 2025)	silent semantic failures evade localization; dominant causes remain non-termination and missing verification (Cemri et al., 2025; Zhu et al., 2025)

Synthesis. Across the three axes, the reliable horizon is a property of the model–harness *pair*: in the notation of §2.2, harness interventions act on $\mathcal{G}$ in $S = (\pi_\theta, \mathcal{G}, \mathcal{E})$, shifting the reliable-horizon surface—the boundary of $\mathcal{H}_\alpha(S, Q)$—without changing π_θ. The same model reaches very different horizons depending on the feedback that catches its errors, the state it can carry, and the verification that lets it run unattended. This is why separating model gains from harness gains is an open evaluation problem, and why we treat useful horizon as a systems property of the coupled model, harness, environment, and verifier.

6 Failure Modes and Open Problems

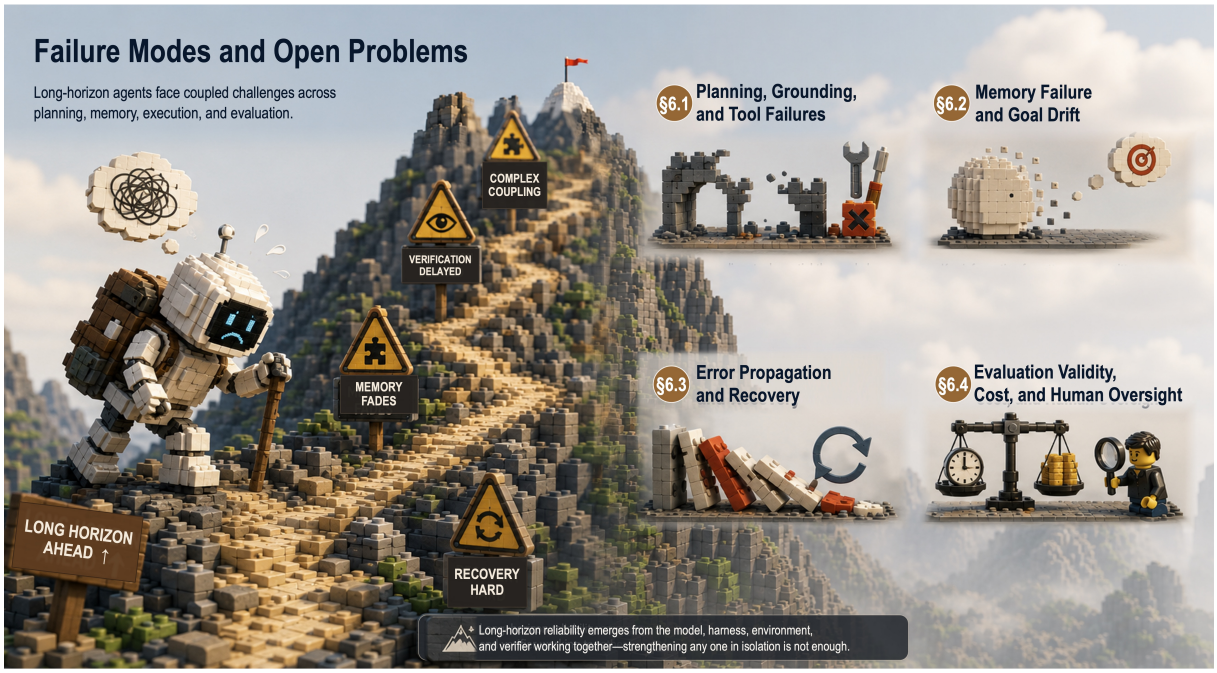

Figure 6: Overview of failure modes and open problems in reliable long-horizon execution. The four groups correspond to the section's analysis of planning, grounding, and tool failures; memory failure and goal drift; error propagation and recovery; and evaluation validity, cost, and human oversight. The central trajectory illustrates how coupled errors, delayed verification, fading memory, and difficult recovery can shorten the reliable horizon.

The preceding sections show that long-horizon agents fail not because one component is missing, but because planning, grounding, memory, tools, verification, cost, and oversight interact across extended trajectories. This section distills the main open problems around the reliable execution horizon: coupled skill failures, memory degradation, error cascades and recovery, attribution, cost, contamination, and human oversight.

6.1 Planning, Grounding, and Tool Failures

Planning, grounding, and tool use are often evaluated as separable capabilities, but long-horizon execution makes them tightly coupled. A correct high-level plan can fail if an agent grounds a browser element, file path, API field, or code location incorrectly; a correctly perceived state can still fail if the agent selects an invalid tool call or violates a domain policy; and a syntactically valid tool call can still be harmful if it advances the wrong subgoal (Deng et al., 2023; Zhou et al., 2024b; Patil et al., 2024; Yao et al., 2024). This coupling explains why benchmarks based on live or simulated environments often reveal failures that are invisible in static reasoning tasks (Xie et al., 2024b; Trivedi et al., 2024a).

A major frontier is therefore to evaluate failure composition rather than isolated skill accuracy. Web and GUI agents must connect language goals to visual or structural observations over changing pages, software agents must map natural-language requirements to cross-file edits and executable tests, and tool agents must satisfy both user intent and hidden state constraints in databases or APIs (Koh et al., 2024; Jimenez et al., 2024; Wang et al., 2024c; Yao et al., 2024). In all cases, a local error can change the future state space: clicking the wrong control, modifying the wrong file, or updating the wrong database row creates new observations that may appear internally consistent while moving the trajectory away from the true goal (Wang et al., 2026c).

Progress requires diagnostic protocols that identify not only whether an agent failed, but which coupling failed first and how the failure propagated. Trajectory-level benchmarks such as AgentBoard and HORIZON move in this direction by emphasizing intermediate states and failure attribution, while stable tool-evaluation infrastructure reduces confounds caused by changing external APIs or nondeterministic graders (Ma et al., 2024; Wang et al., 2026c; Guo et al., 2024). Future evaluations should annotate whether

failures originate in decomposition, observation grounding, tool selection, argument construction, policy compliance, state tracking, or verification, because each source implies a different model or harness intervention (Rabanser et al., 2026).

6.2 Memory Failure and Goal Drift

Long-horizon agents must decide what to remember, what to summarize, what to retrieve, what to discard, and what to distrust. Memory is therefore not merely a larger context window: it is a state-management mechanism that must preserve user constraints, intermediate artifacts, tool outputs, environment observations, and unresolved subgoals while preventing stale or false information from contaminating future decisions (Packer et al., 2023; Gutiérrez et al., 2024; Hu et al., 2025b; Du, 2026). This distinction matters because a memory system that stores more text can still shorten the reliable horizon if it retrieves outdated state or compresses away task-critical constraints (Mei et al., 2025; Wu et al., 2026b).

Goal drift arises when an agent's active objective gradually diverges from the user's original intent or from the current environment state. It can occur through lossy summaries, irrelevant retrieval, repeated self-generated assumptions, failed attempts copied into scratchpads, or over-optimization of local subgoals such as passing a test while violating a broader specification (Shinn et al., 2023; Shi et al., 2025; Yan et al., 2026). In coding and workflow tasks, this drift is especially dangerous because intermediate artifacts become part of the world: a wrong edit, invalid cached result, or obsolete plan may be repeatedly reinforced by later observations (Jimenez et al., 2024; Wang et al., 2026c).

The research frontier is to evaluate memory by *state fidelity* and *action usefulness*, not by storage capacity alone. Useful memory should preserve task-critical invariants, identify the freshness and provenance of stored facts, support contradiction detection, and expose uncertainty when retrieved information may no longer match the environment (Xu et al., 2026d; Du, 2026; Mei et al., 2025). Harnesses can help by separating durable user constraints from temporary observations, attaching timestamps and tool provenance to memory entries, maintaining checkpoints, and forcing revalidation before irreversible actions (Qiao et al., 2023; Wang et al., 2025d; Mozannar et al., 2025).

6.3 Error Propagation and Recovery

Long-horizon language agents do not make decisions from a stateless context. Each step is conditioned on an accumulated trajectory of prior actions, observations, tool responses, intermediate plans, and, in some systems, persistent memory. As a result, an early mistake may not simply cause a local failure; it can be incorporated into the agent state and subsequently shape later reasoning and action selection. This phenomenon gives rise to **error cascades**, where downstream decisions may appear coherent with the recorded trajectory while still being causally downstream of an earlier fault. Recent diagnostic studies make this issue explicit: TOOLSCAN (Kokane et al., 2025) moves beyond final success rate by characterizing fine-grained tool-use errors, such as invalid formats, hallucinated functions, incorrect arguments, repeated calls, and insufficient API calls, while AgentErrorTaxonomy (Zhu et al., 2025) attributes failures to memory, reflection, planning, action, and system-level modules and highlights error propagation as a central bottleneck in agent reliability.

A common mitigation strategy is to introduce iterative critique and revision. Self-Refine (Madaan et al., 2023) turns generation into a feedback and refinement loop in which the same model critiques and revises its own output, whereas Reflexion (Shinn et al., 2023) stores verbal reflections from failed trials as episodic memory to guide subsequent attempts. These methods show that feedback-conditioned revision can improve agent behavior in many settings. However, their success should not be conflated with reliable intrinsic self-correction. In reasoning tasks, models often fail to judge the correctness of their own outputs without additional evidence, and self-correction can even reduce performance (Huang et al., 2024). This suggests that recovery is less a matter of simply generating another response than of identifying which part of the previous trajectory should be revised, ignored, or discarded.

External feedback provides a more grounded basis for such diagnosis. Tool outputs, execution traces, verifier judgments, simulator observations, and environment errors can expose discrepancies between an agent's intended plan and the actual state of the world. CRITIC (Gou et al., 2024) illustrates this principle by verifying model outputs through external tools and using the resulting critiques to guide

revision. In agentic planning, Hell or High Water (Wang et al., 2025a) isolates a complementary setting: agents encounter external tool failures while the task remains solvable through alternative function compositions, thereby testing whether they can abandon a failed plan and construct a backup strategy. PALADIN (Vuddanti et al., 2025) further treats recovery as a learnable execution-level behavior: it injects tool failures into ToolBench-style trajectories, annotates recovery actions, fine-tunes agents on failure-rich rollouts, and retrieves taxonomy-aligned recovery exemplars at inference time to guide retry, tool substitution, replanning, or graceful termination. These studies indicate that recovery requires not only retrying, but also interpreting feedback, localizing the failure source, and adapting the subsequent action space.

In summary, this line of work reframes agent robustness as a state-management problem. A recovery protocol must specify the failure type, the observability of the failure signal, whether the task remains feasible, what alternative actions are available, and how much of the accumulated trajectory should continue to be trusted. Open challenges remain in distinguishing recoverable external failures from invalid tasks, detecting silent semantic failures that produce plausible but wrong observations, and deciding when persistence should give way to safe abstention. Error cascades, self-correction, and recovery should therefore be studied jointly, as mechanisms for controlling how unreliable trajectory state is diagnosed, revised, and propagated.

6.4 Evaluation Validity, Cost, and Human Oversight

Method-level progress alone does not determine whether language agents are reliable. Modern agents are increasingly deployed as systems: their behavior is shaped not only by the base model, but also by context assembly, memory retrieval, tool interfaces, search and verification policies, execution controllers, and human intervention. Recent surveys on context engineering, self-evolving agents, and agent memory make this systems view explicit, treating context, memory, tools, and adaptation as first-class determinants of agent behavior rather than auxiliary prompting choices (Mei et al., 2025; Gao et al., 2026; Hu et al., 2025b; Du, 2026). This shift raises a methodological question that current benchmarks only partially address: when an agent succeeds, what exactly should the success be attributed to, under what resource budget, with what contamination risk, and under what autonomy constraints?

Model–harness attribution. A central challenge is to disentangle model capability from harness effects. Improvements on agent benchmarks may reflect a stronger base model, but they may also arise from better context construction, memory retrieval, tool schemas, search depth, verifier design, retry policies, or interface constraints. Final task success is therefore insufficient evidence of model-level progress. More rigorous evaluation should report both model-only and system-level performance, and should include controlled ablations that vary one component while holding the others fixed. This requires, for example, freezing the harness while changing the model, freezing the model while changing memory or tool policies, and reporting cross-combinations of models and scaffolds. Trajectory-centric benchmarks such as AgentBoard move in this direction by evaluating agents through multi-step action–observation processes rather than single-shot outputs (Ma et al., 2024). However, the field still lacks standardized factorial protocols for attributing gains across model, memory, tool, verifier, and execution-policy components.

Cost-normalized evaluation. Agent reliability should also be compared under explicit resource budgets. Higher success rates can often be obtained by increasing prompt length, sampling more candidates, deepening search, adding reflection rounds, calling more tools, using stronger verifiers, or coordinating larger agent teams. Without cost normalization, benchmark comparisons may conflate capability with expenditure. Recent work on scaling agent systems shows that coordination benefits are highly task-dependent: multi-agent systems can help on decomposable tasks but can also incur coordination overhead, saturation effects, and topology-dependent error amplification on sequential or tool-heavy tasks (Kim et al., 2026). Evaluation should therefore report success jointly with token usage, wall-clock latency, number of model calls, tool-call counts, verifier cost, memory growth, and human-intervention cost. In many settings, Pareto frontiers over accuracy, cost, and latency are more informative than a single aggregate success score.

Benchmark contamination. Contamination is particularly difficult to control in agentic evaluation. Leakage may occur not only through final answers, but also through task templates, tool schemas, website states, demonstrations, grading scripts, public solution traces, recovery exemplars, or benchmark-specific prompting conventions. Thus, strong performance on a long-horizon benchmark may reflect familiarity with the evaluation environment rather than robust generalization. General studies of benchmark data contamination distinguish multiple levels of exposure, from semantic and information-level leakage to direct data and label leakage (Xu et al., 2024a). Dynamic benchmarks provide complementary mitigation strategies: LatestEval (Li et al., 2024) constructs reading-comprehension tests from recent texts, LiveCodeBench (Jain et al., 2025) uses time-stamped programming problems and evaluates models on post-cutoff data, and LiveBench (White et al., 2024) combines frequently updated questions with objective ground-truth scoring. For agents, these ideas need to be extended beyond fresh questions to regenerated environments, rotating tool schemas, hidden tool states, private task distributions, and trace-level leakage audits.

Adaptive autonomy and human oversight. A further frontier is determining when agents should act autonomously and when they should defer to human judgment. Oversight should not be reduced to a binary choice between full automation and manual control. Depending on risk and reversibility, an agent may need to request clarification, seek approval before irreversible actions, checkpoint its state, roll back a previous step, abstain under uncertainty, or escalate to a human operator. This issue is acute in tool-using and state-changing environments, where errors can affect external systems rather than merely produce incorrect text. Safety benchmarks such as AgentDojo (Debenedetti et al., 2024) and Agent-SafetyBench (Zhang et al., 2024) show that tool-integrated agents introduce behavior-level risks, including unsafe tool invocation, prompt-injection vulnerability, excessive trust in tool outputs, and insufficient risk awareness. Governance-oriented work further emphasizes constrained action spaces, approval gates, audit trails, sandboxing, rollback mechanisms, and accountability for deployed agentic systems (Shavit et al., 2023). Future protocols should treat human intervention as a scarce but necessary resource, reporting escalation precision, approval burden, intervention latency, rollback success, reversibility, and post-hoc trace interpretability.

In summary, these frontiers shift the unit of analysis from isolated task success to the quality of evidence supporting a reliability claim. A mature evaluation protocol should disclose the base model, harness configuration, context policy, memory policy, tool budget, verifier budget, contamination controls, autonomy policy, and human-intervention interface. Without such reporting, progress remains difficult to attribute, expensive to reproduce, and unsafe to extrapolate to open-ended deployment.

7 Conclusion and Outlook

LLM agents are increasingly expected to solve tasks that require extended interaction with tools, software environments, web interfaces, databases, and other mutable external states. The central question is not whether a system can occasionally complete a long task, but whether it remains reliably delegable as task pressure increases. This survey separates six task-pressure axes from the outcome stack, represents reliable horizon as a protocol-dependent surface, and interprets scalar horizons only as declared conditional slices. On this view, model or harness progress is established by a controlled outward shift of the reliable boundary under matched tasks, budgets, and verification—not by a larger action count or a single successful trajectory. Reliable horizon is therefore a measurable property of the full model–harness–environment–evaluation system.

References

Petr Anokhin, Roman Khalikov, Stefan Rebrikov, Viktor Volkov, Artyom Sorokin, and Vincent Bissonnette. 2025. Herobench: A benchmark for long-horizon planning and structured reasoning in virtual worlds. *arXiv preprint arXiv:2508.12782*.

Sagnik Anupam, Davis Brown, Shuo Li, Eric Wong, Hamed Hassani, and Osbert Bastani. 2025. Browserarena: Evaluating llm agents on real-world web navigation tasks. *arXiv preprint arXiv:2510.02418*.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. 2026. Swe-rebench v2: Language-agnostic swe task collection at scale. *arXiv preprint arXiv:2602.23866*.

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

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, and 1 others. 2025. Windows agent arena: Evaluating multi-modal os agents at scale. In *International Conference on Machine Learning*, pages 4874–4910. PMLR.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A Zaharia, Joseph Gonzalez, and Ion Stoica. 2025. Why do multi-agent llm systems fail? In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc.

Guoxin Chen, Zile Qiao, Xuanzhong Chen, Donglei Yu, Haotian Xu, Wayne Xin Zhao, Ruihua Song, Wenbiao Yin, Huifeng Yin, Liwen Zhang, and 1 others. 2025a. Iterresearch: Rethinking long-horizon agents with interaction scaling. *arXiv preprint arXiv:2511.07327*.

Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. 2026. Swe-ci: Evaluating agent capabilities in maintaining codebases via continuous integration. *Preprint*, arXiv:2603.03823.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. 2025b. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *International Conference on Learning Representations*, volume 2024, pages 8746–8825.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Sanjiban Choudhury. 2025. Process reward models for llm agents: Practical framework and directions. *Preprint*, arXiv:2502.10325.

Evan Chu, Rajan Agarwal, Abishek Thangamuthu, Brendan Graham, Justus Mattern, Freeman Jiang, Paul Cento, Swarnim Jain, Mersad Abbasi, Mohammad Hossein Rezaei, George Wang, Alex Zhang, Simon Guo, Karina Nguyen, Danna Liu, Arash Bidgoli, Aditya Dalmia, Apoorv Dankar, Ashrut Vaddela, and 12 others. 2026. Frontierswe. *Proximal Blog*. Https://frontierswe.com/blog.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in Neural Information Processing Systems*, 37:82895–82920.

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. 2026. SWE-Milestone: Evaluating AI agents on continuous software evolution. *Preprint*, arXiv:2603.13428.

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, and 3 others. 2025. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *Preprint*, arXiv:2509.16941.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Rishi Desai, Jesse Hu, Joan Cabezas, Neel Harsola, Pratyush Shukla, Roey Ben Chaim, Adnan El Assadi, Omkaar Mukund Kamath, Fenil Faldu, Prannay Hebbar, Jiankai Sun, Yiyuan Li, Pramod Srinivasan, Ishan Gupta, Christopher Settles, Daniel Wang, Derek Chen, Pranav Raja, Albert Liu, and 7 others. 2026. Swe-marathon: Can agents autonomously complete ultra-long-horizon software work? *Preprint*, arXiv:2606.07682.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2024. Chain-of-verification reduces hallucination in large language models. In *Findings of the association for computational linguistics: ACL 2024*, pages 3563–3578.

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, and 30 others. 2026. Nl2repo-bench: Towards long-horizon repository generation evaluation of coding agents. *Preprint*, arXiv:2512.12730.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, and 1 others. 2024. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. 2025. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*.

Pengfei Du. 2026. Memory for autonomous llm agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2026. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 17490–17502, San Diego, California, United States. Association for Computational Linguistics.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2026a. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408.

Yukang Feng, Jianwen Sun, Zelai Yang, Jiaxin Ai, Chuanhao Li, Zizhen Li, Fanrui Zhang, Kang He, Rui Ma, Jifan Lin, Jie Sun, Yang Xiao, Sizhuo Zhou, Wenxiao Wu, Yiming Liu, Pengfei Liu, Yu Qiao, Shenglin Zhang, and Kaipeng Zhang. 2026b. Longcli-bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces. *Preprint*, arXiv:2602.14337.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, and 1 others. 2026. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*.

Gonzalo Gonzalez-Pumariega, Leong Yean, Neha Sunkara, and Sanjiban Choudhury. 2025. Robotouille: An asynchronous planning benchmark for llm agents. In *International Conference on Learning Representations*, volume 2025, pages 83757–83792.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jiménez Gutiérrez, Yiheng Shu, and 1 others. 2025. Mind2web 2: Evaluating agentic search with agent-as-a-judge, 2025. *URL https://arxiv. org/abs/2506.21506*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, and 1 others. 2024. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pages 57734–57811.

Hao Guan, Lingyue Fu, Shao Zhang, Yaoming Zhu, Kangning Zhang, Lin Qiu, Xunliang Cai, Xuezhi Cao, Weiwen Liu, Weinan Zhang, and Yong Yu. 2026. Swe-cycle: Benchmarking code agents across the complete issue resolution cycle. *Preprint*, arXiv:2605.13139.

Anmol Gulati, Hariom Gupta, Elias Lumer, Sahil Sen, and Vamse Kumar Subbiah. 2026. Ask early, ask late, ask right: When does clarification timing matter for long-horizon agents? *Preprint*, arXiv:2605.07937.

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

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569.

Songhao Han, Boxiang Qiu, Yue Liao, Siyuan Huang, Chen Gao, Shuicheng Yan, and Si Liu. 2026. Robocerebra: A large-scale benchmark for long-horizon robotic manipulation evaluation. *Advances in Neural Information Processing Systems*, 38.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890.

William Hill, Ireton Liu, Anita De Mello Koch, Damion Harvey, Nishanth Kumar, George Konidaris, and Steven James. 2023. Mineplanner: A benchmark for long-horizon planning in large minecraft worlds. *arXiv preprint arXiv:2312.12891*.

Jiseung Hong, Benjamin G. Ascoli, and Jinho D. Choi. 2026. Recube: Evaluating repository-level context utilization in code generation. *Preprint*, arXiv:2603.25770.

Chen Hu, Haikuo Du, Heng Wang, Lin Lin, Mingrui Chen, Peng Liu, Ruihang Miao, Tianchi Yue, Wang You, Wei Ji, and 1 others. 2025a. Step-deepresearch technical report. *arXiv preprint arXiv:2512.20491*.

Ruida Hu, Xinchen Wang, Chao Peng, Cuiyun Gao, and David Lo. 2026. Evaluating llm-based 0-to-1 software generation in end-to-end cli tool scenarios. *Preprint*, arXiv:2604.06742.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, and 1 others. 2025b. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pages 32808–32824.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and brian ichter. 2023. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. 2026. Deepswe: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. 2026. Ale-bench: A benchmark for long-horizon objective-driven algorithm engineering. *Advances in Neural Information Processing Systems*, 38.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

Qirui Jin, Lingching Tung, Kenan Li, Qiyang Shi, Yushi She, Huanzhong Jia, Harrison Zhao, Kejing Xia, Zhenbang Du, Yikai Zhang, and 1 others. 2026a. Chainswe: Benchmarking coding agents on multi-bug software maintenance. *arXiv preprint arXiv:2607.02606*.

Yiyang Jin, Kunzhao Xu, Hang Li, Xueting Han, Yanmin Zhou, Cheng Li, and Jing Bai. 2026b. Reveal: Self-evolving code agents via reliable self-verification. In *The Fourteenth International Conference on Learning Representations*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. 2025. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. 2024. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pages 161–178. Springer.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, and 1 others. 2025. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. *arXiv preprint arXiv:2510.11977*.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. 2026. Process reward models that think. *Transactions on Machine Learning Research*. J2C Certification.

Aaditya Khanal, Yangyang Tao, and Junxiu Zhou. 2026. Beyond pass@ 1: A reliability science framework for long-horizon llm agents. *arXiv preprint arXiv:2603.29231*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. 2026. Towards a science of scaling agent systems. *Preprint*, arXiv:2512.08296.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.

Shirley Kokane, Ming Zhu, Tulika Manoj Awalgaonkar, Jianguo Zhang, Akshara Prabhakar, Thai Quoc Hoang, Zuxin Liu, Rithesh RN, Liangwei Yang, Weiran Yao, and 1 others. 2025. Toolscan: A benchmark for characterizing errors in tool-use llms. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

Ádám Kovács. 2026. Squeez: Task-conditioned tool-output pruning for coding agents. *arXiv preprint arXiv:2604.04979*.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, and 1 others. 2026. Measuring ai ability to complete long software tasks. *Advances in Neural Information Processing Systems*, 38:92213–92266.

Man Ho Lam, Chaozheng Wang, Hange Liu, Jingyu Xiao, Haau sing Li, Jen tse Huang, Terry Yue Zhuo, and Michael R. Lyu. 2026. Swe-chain: Benchmarking coding agents on chained release-level package upgrades. *Preprint*, arXiv:2605.14415.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. 2026. Swe-evo: Benchmarking coding agents in long-horizon software evolution scenarios. *Preprint*, arXiv:2512.18470.

Dongjun Lee, Juyong Lee, Kyuyoung Kim, Jihoon Tack, Jinwoo Shin, Yee Whye Teh, and Kimin Lee. 2025. Learning to contextualize web pages for enhanced decision making by LLM agents. In *The Thirteenth International Conference on Learning Representations*.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. 2024. Benchmarking mobile device control agents across diverse configurations. *arXiv preprint arXiv:2404.16660*.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, Nir Mashkif, and Segev Shlomov. 2024. St-webagentbench: A benchmark for evaluating safety and trustworthiness in web agents. *arXiv preprint arXiv:2410.06703*.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, and 1 others. 2023. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR.

Chenliang Li, Adel Elmahdy, Alex Boyd, Zhongruo Wang, Siliang Zeng, Alfredo Garcia, Parminder Bhatia, Taha Kass-Hout, Cao Xiao, and Mingyi Hong. 2025a. Stabilizing off-policy training for long-horizon llm agent via turn-level importance sampling and clipping-triggered normalization. *arXiv preprint arXiv:2511.20718*.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. 2026a. Salt: Step-level advantage assignment for long-horizon agents via trajectory graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4709–4725.

Wanli Li, Bowen Zhou, Yunyao Yu, Zhou Xu, Yifan Yang, Dongsheng Li, and Caihua Shan. 2026b. Weavebench: A long-horizon, real-world benchmark for computer-use agents with hybrid interfaces. *arXiv preprint arXiv:2606.09426*.

Yucheng Li, Frank Guerin, and Chenghua Lin. 2024. Latesteval: Addressing data contamination in language model evaluation through dynamic and time-sensitive test construction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18600–18607.

Zijian Li, Xin Guan, Bo Zhang, Shen Huang, Houquan Zhou, Shaopeng Lai, Ming Yan, Yong Jiang, Pengjun Xie, Fei Huang, and 1 others. 2025b. Webweaver: Structuring web-scale evidence with dynamic outlines for open-ended deep research. *arXiv preprint arXiv:2509.13312*.

Tobias Lindenbauer, Igor Slinko, Ludwig Felder, Egor Bogomolov, and Yaroslav Zharov. 2025. The complexity trap: Simple observation masking is as efficient as llm summarization for agent context management. *arXiv preprint arXiv:2508.21433*.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. 2026a. Dive into claude code: The design space of today's and future ai agent systems. *Preprint*, arXiv:2604.14228.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2024. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Yue Liu, Yingwei Ma, Yibo Miao, Yanhao Li, Yuchong Xie, Xinlong Yang, Zhiyuan Hu, Flood Sung, Jiaheng Zhang, and Bryan Hooi. 2026b. Klong: Training llm agent for extremely long-horizon tasks. *arXiv preprint arXiv:2602.17547*.

Qingyu Lu, Liang Ding, Siyi Cao, Xuebo Liu, Kanjian Zhang, Jinxia Zhang, and Dacheng Tao. 2025. Runaway is ashamed, but helpful: On the early-exit behavior of large language model-based agents in embodied environments. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 24014–24027, Suzhou, China. Association for Computational Linguistics.

Xing Han Lù, Zdenek Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 3(8).

Haotian Luo, Huaisong Zhang, Xuelin Zhang, Haoyu Wang, Zeyu Qin, Wenjie Lu, Guozheng Ma, Haiying He, Yingsha Xie, Qiyang Zhou, and 1 others. 2025. Ultrahorizon: Benchmarking agent capabilities in ultra long-horizon scenarios. *arXiv preprint arXiv:2509.21766*.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2025. Dover: Intervention-driven auto debugging for llm multi-agent systems. *arXiv preprint arXiv:2512.06749*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. 2024. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, and 1 others. 2025. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*.

Qianyu Meng, Yanan Wang, Liyi Chen, Qimeng Wang, Chengqiang Lu, Wei Wu, Yan Gao, Yi Wu, and Yao Hu. 2026. Agent harness for large language model agents: A survey. *Preprints*.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, and 66 others. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *Preprint*, arXiv:2601.11868.

Elliot Meyerson, Giuseppe Paolo, Roberto Dailey, Hormoz Shahrzad, Olivier Francon, Conor F Hayes, Xin Qiu, Babak Hodjat, and Risto Miikkulainen. 2025. Solving a million-step llm task with zero errors. *arXiv preprint arXiv:2511.09030*.

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. 2025. Evaluation and benchmarking of llm agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 6129–6139.

Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, Eric Zhu, Griffin Bassman, Jacob Alber, Peter Chang, Ricky Loynd, Friederike Niedtner, Ece Kamar, Maya Murad, Rafah Hosn, and Saleema Amershi. 2025. Magentic-ui: Towards human-in-the-loop agentic systems. *Preprint*, arXiv:2507.22358.

Rabeya Khatun Muna, Md Nakhla Rafi, Tse-Hsun, and Chen. 2026. Ci-repair-bench: A repository-aware benchmark for automated patch validation via ci workflows. *Preprint*, arXiv:2604.27148.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2022. Webgpt: Browser-assisted question-answering with human feedback, 2022. *URL https://arxiv. org/abs/2112.09332*, 35.

OpenAI. 2024. Introducing swe-bench verified.

Toby Ord. 2025. Is there a half-life for the success rates of ai agents? *arXiv preprint arXiv:2505.05115*.

Yuki Orimo, Iori Kurata, Hodaka Mori, Ryuhei Okuno, Ryohto Sawada, and Daisuke Okanohara. 2025. Parc: An autonomous self-reflective coding agent for robust execution of long-horizon tasks. *Preprint*, arXiv:2512.03549.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and 1 others. 2024. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

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

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Adamenko Pavel, Ivanov Mikhail, Aidar Valeev, Rodion Levichev, Pavel Zadorozhny, Ivan Lopatin, Dmitrii Babaev, Alena Fenogenova, and Valentin Malykh. 2025. Swe-mera: A dynamic benchmark for agenticly evaluating large language models on software engineering tasks. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 440–452.

Zhiyuan Peng, Xin Yin, Pu Zhao, Fangkai Yang, Lu Wang, Ran Jia, Xu Chen, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2026. Repogenesis: Benchmarking end-to-end microservice generation from readme to repository. *Preprint*, arXiv:2601.13943.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, and 1 others. 2023. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. 2026. Towards a science of ai agent reliability. *arXiv preprint arXiv:2602.16666*.

Mohit Raghavendra, Soham Dan, Miguel Romero Calvo, Yannis Yiming He, Johannes Baptist Mols, Gautam Anand, Cole McCollum, Edgar Arakelyan, Vijay Bharadwaj, Andrew Park, Jeff Da, MohammadHossein Rezaei, Bing Liu, Brad Kenstler, and Yunzhong He. 2026. Swe atlas: Benchmarking coding agents beyond issue resolution. *Preprint*, arXiv:2605.08366.

Guruprasad Viswanathan Ramesh, Asmit Nayak, Basieem Siddique, and Kassem Fawaz. 2026. Websp-eval: Evaluating web agents on website security and privacy tasks. *arXiv preprint arXiv:2604.06367*.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. 2026. Posttrainbench: Can llm agents automate llm post-training? *Preprint*, arXiv:2603.08640.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, and 1 others. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pages 406–441.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Ye Shang, Quanjun Zhang, Haichuan Hu, Chunrong Fang, Liang Xiao, and Zhenyu Chen. 2026. Breaking, stale, or missing? benchmarking coding agents on project-level test evolution. *Preprint*, arXiv:2605.06125.

Yonadav Shavit, Sandhini Agarwal, Miles Brundage, Steven Adler, Cullen O'Keefe, Rosie Campbell, Teddy Lee, Pamela Mishkin, Tyna Eloundou, Alan Hickey, and 1 others. 2023. Practices for governing agentic ai systems. *Research Paper, OpenAI*.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR.

Yuling Shi, Yichun Qian, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2025. Longcodezip: Compress long context for code language models. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 141–153.

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

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. 2025. The illusion of diminishing returns: Measuring long horizon execution in llms. *arXiv preprint arXiv:2509.09677*.

{Theodore R.} Sumers, Shunyu Yao, Karthik Narasimhan, and {Thomas L.} Griffiths. 2024. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. Publisher Copyright: © 2024, Transactions on Machine Learning Research. All rights reserved.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*.

Zihao Sun and Ling Chen. 2025. Webarxiv: Evaluating multimodal agents on time-invariant arxiv tasks. *arXiv preprint arXiv:2507.00938*.

Chris Sypherd and Vaishak Belle. 2024. Practical considerations for agentic llm systems. *Preprint*, arXiv:2412.04093.

Hui-Ze Tan, Xiao-Wen Yang, Hao Chen, Jie-Jing Shao, Yi Wen, Yuteng Shen, Weihong Luo, Xiku Du, Lan-Zhe Guo, and Yu-Feng Li. 2026. Hindsight credit assignment for long-horizon llm agents. *arXiv preprint arXiv:2603.08754*.

Natalia Tarasova, Enrique Balp-Straffon, Aleksei Iancheruk, Yevhenii Sielskyi, Nikita Kozodoi, Liam H. Byrne, Jack Butler, Dayuan Jiang, Marcin Czelej, Andrew Ang, Yash Shah, Roi Blanco, and Sergei Ivanov. 2026. Swe-infrabench: Evaluating language models on cloud infrastructure code. *Preprint*, arXiv:2606.05249.

George Thomas, Alex J Chan, Jikun Kang, Wenqi Wu, Filippos Christianos, Fraser Greenlee, Andy Toulis, and Marvin Purtorab. 2025. Webgames: Challenging general-purpose web-browsing ai agents. *arXiv preprint arXiv:2502.18356*.

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

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

Ada Defne Tur, Nicholas Meade, Xing Han Lù, Alejandra Zambrano, Arkil Patel, Esin Durmus, Spandana Gella, Karolina Stańczak, and Siva Reddy. 2025. Safearena: Evaluating the safety of autonomous web agents. *arXiv preprint arXiv:2503.04957*.

Sri Vatsa Vuddanti, Aarav Shah, Satwik Kumar Chittiprolu, Tony Song, Sunishchal Dev, Kevin Zhu, and Maheep Chaudhary. 2025. Paladin: Self-correcting language model agents to cure tool-failure cases. *arXiv preprint arXiv:2509.25238*.

Andrew Wang, Sophia Hager, Adi Asija, Daniel Khashabi, and Nicholas Andrews. 2025a. Hell or high water: Evaluating agentic recovery from external failures. In *Second Conference on Language Modeling*.

Binghai Wang, Chenlong Zhang, Dayiheng Liu, Jiajun Zhang, Jiawei Chen, Mingze Li, Mouxiang Chen, Rongyao Fang, Siyuan Zhang, Xuwu Wang, Yuheng Jing, Zeyao Ma, and Zeyu Cui. 2026a. The verification horizon: No silver bullet for coding agent rewards. *Preprint*, arXiv:2606.26300.

Daoyu Wang, Qingchuan Li, Mingyue Cheng, Jie Ouyang, Shuo Yu, Qi Liu, and Enhong Chen. 2026b. Steppo: Step-aligned policy optimization for agentic reinforcement learning. *arXiv preprint arXiv:2604.18401*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Haoran Wang, Zhenyu Hou, Yao Wei, Jie Tang, and Yuxiao Dong. 2025b. SWE-dev: Building software engineering agents with training and inference scaling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3742–3761, Vienna, Austria. Association for Computational Linguistics.

Maria Wang, Srinivas Sunkara, Gilles Baechler, Jason Lin, Yun Zhu, Fedir Zubach, Lei Shu, and Jindong Chen. 2024b. Webquest: A benchmark for multimodal qa on web page sequences. corr, abs/2409.13711, 2024. doi: 10.48550. *arXiv preprint arXiv:2409.13711*.

Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. 2025c. Odysseybench: Evaluating llm agents on long-horizon complex office application workflows. *arXiv preprint arXiv:2508.09124*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024c. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2025d. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pages 65882–65919.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024d. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, volume 2024, pages 32593–32627.

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

Yifei Wang, Ziteng Wang, Yuling Shi, Silin Chen, Xinrui Wang, Yueqi Wang, Beijun Shen, Linjing Li, Xiaodong Gu, Julian McAuley, and Daniel Dajun Zeng. 2026d. Context compression for llm agents: A survey of methods, failure modes, and evaluation. *Preprints*.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. 2026e. Swe-pruner: Self-adaptive context pruning for coding agents. *arXiv preprint arXiv:2601.16746*.

Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025e. Agent workflow memory.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. 2025f. JARVIS-1: Open-World Multi-Task Agents With Memory-Augmented Multimodal Language Models . *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 47(03):1894–1907.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. 2024e. Officebench: Benchmarking language agents across multiple applications for office automation. *arXiv preprint arXiv:2407.19056*.

Hu Wei. 2026. From agent loops to structured graphs:a scheduler-theoretic framework for llm agent execution. *Preprint*, arXiv:2604.11378.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, and 1 others. 2024. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 4:2.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, and 1 others. 2024. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, and 1 others. 2025. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*.

Yating Wu, Yuhao Zhang, Sayan Ghosh, Sourya Basu, Anoop Deoras, Jun Huan, and Gaurav Gupta. 2026a. Contextweaver: Selective and dependency-structured memory construction for llm agents. *arXiv preprint arXiv:2604.23069*.

Yong Wu, YanZhao Zheng, TianZe Xu, ZhenTao Zhang, YuanQiang Yu, JiHuai Zhu, Chao Ma, BinBin Lin, BaoHua Dong, HangCheng Zhu, and 1 others. 2026b. Contextbudget: Budget-aware context management for long-horizon search agents. *arXiv preprint arXiv:2604.01664*.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Agentless: Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE).

Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. 2026. Reducing cost of llm agents with trajectory reduction. *Proceedings of the ACM on Software Engineering*, 3(FSE):1241–1263.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024a. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024b. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.

Cheng Xu, Shuhao Guan, Derek Greene, M Kechadi, and 1 others. 2024a. Benchmark data contamination of large language models: A survey. *arXiv preprint arXiv:2406.04244*.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, and 1 others. 2026a. Theagentcompany: benchmarking llm agents on consequential real world tasks. *Advances in Neural Information Processing Systems*, 38.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, and 1 others. 2026b. Theagentcompany: benchmarking llm agents on consequential real world tasks. *Advances in Neural Information Processing Systems*, 38.

Huiyu Xu, Zhibo Wang, Wenhui Zhang, Ziqi Zhu, Yaopeng Wang, Kui Ren, and Chun Chen. 2026c. Looptrap: Termination poisoning attacks on llm agents. *Preprint*, arXiv:2605.05846.

Kevin Xu, Yeganeh Kordi, Tanay Nayak, Adi Asija, Yizhong Wang, Kate Sanders, Adam Byerly, Jingyu Zhang, Benjamin Van Durme, and Daniel Khashabi. 2024b. Tur[k]ingbench: A challenge benchmark for web agents. *arXiv preprint arXiv:2403.11905*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026d. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604.

Xinbo Xu, Ruihan Yang, Haiyang Shen, Wendong Xu, Bofei Gao, Ruoyu Wu, Kean Shi, Weichu Xie, Xuanzhong Chen, Ming Wu, Jason Zeng, Michael Heinrich, Elvis Zhang, Liang Chen, Kuan Li, and Baobao Chang. 2026e. Roadmapbench: Evaluating long-horizon agentic software development across version upgrades. *Preprint*, arXiv:2605.15846.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. An illusion of progress? assessing the current state of web agents, 2025. *URL https://arxiv. org/abs/2504.01382*.

Lu Yan, Xuan Chen, and Xiangyu Zhang. 2026. When the specification emerges: Benchmarking faithfulness loss in long-horizon coding agents. *arXiv preprint arXiv:2603.17104*.

Chengyuan Yang, Zequn Sun, Wei Wei, and Wei Hu. 2026a. Beyond static summarization: Proactive memory extraction for llm agents. *arXiv preprint arXiv:2601.04463*.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. 2026b. Programbench: Can language models rebuild programs from scratch? *Preprint*, arXiv:2605.03546.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023b. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023c. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, and 1 others. 2025. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. A survey on evaluation of llm-based agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 26690–26714.

Zonghao Ying, Yangguang Shao, Jianle Gan, Gan Xu, Wenxin Zhang, Quanchen Zou, Junzheng Shi, Zhenfei Yin, Mingchuan Zhang, Aishan Liu, and 1 others. 2026. Securewebarena: A holistic security evaluation benchmark for lvlm-based web agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 11986–11998.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. 2024. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8938–8968.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, and 17 others. 2026. OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks. *Preprint*, arXiv:2606.29537.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, and 1 others. 2026a. Swe-bench goes live! *Advances in Neural Information Processing Systems*, 38.

Linghao Zhang, Junhao Wang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Jiaheng Wen, Chengxing Xie, Maoquan Wang, Yufan Huang, and 1 others. 2025a. Di-bench: Benchmarking large language models on dependency inference with testable repositories at scale. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10134–10153.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026b. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Shaoqiu Zhang, Yuhang Wang, Jialiang Liang, Yuling Shi, Wenhao Zeng, Maoquan Wang, Shilin He, Ningyuan Xu, Siyu Ye, Kai Cai, and Xiaodong Gu. 2026c. Swe-explore: Benchmarking how coding agents explore repositories. *Preprint*, arXiv:2606.07297.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. 2026d. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints. *arXiv preprint arXiv:2601.18137*.

Zehua Zhang, Ati Priya Bajaj, Divij Handa, Siyu Liu, Arvind S Raj, Hongkai Chen, Hulin Wang, Yibo Liu, Zion Leonahenahe Basque, Souradip Nath, Vishal Juneja, Nikhil Chapre, Yan Shoshitaishvili, Adam Doupé, Chitta Baral, and Ruoyu Wang. 2025b. Buildbench: Benchmarking llm agents on compiling real-world open-source software. *Preprint*, arXiv:2509.25248.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Bingchen Zhao, Dhruv Srikanth, Yuxiang Wu, and Zhengyao Jiang. 2026a. Specbench: Measuring reward hacking in long-horizon coding agents. *Preprint*, arXiv:2605.21384.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, and 1 others. 2026b. Ama-bench: Evaluating long-horizon memory for agentic applications. *arXiv preprint arXiv:2602.22769*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, and 1 others. 2024b. Natural plan: Benchmarking llms on natural language planning. *arXiv preprint arXiv:2406.04520*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and {Yu Xiong} Wang. 2024a. Language agent tree search unifies reasoning, acting, and planning in language models. *Proceedings of Machine Learning Research*, 235:62138–62160. We thank Daniel Campos for useful feedback on earlier versions of this paper.This work was supported in part by NSF Grant 2106825, NIFA Award 2020-67021-32799, the Jump ARCHES endowment through the Health Care Engineering Systems Center at Illinois and the OSF Foundation, and the IBM-Illinois Discovery Accelerator Institute.This work used NVIDIA GPUs at NCSA Delta through allocations CIS220014, CIS230012, and CIS230218 from the ACCESS program.; 41st International Conference on Machine Learning, ICML 2024 ; Conference date: 21-07-2024 Through 27-07-2024.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, and 1 others. 2026a. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*.

Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, and 1 others. 2025a. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *arXiv preprint arXiv:2504.19314*.

Qixing Zhou, Jiacheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. 2026b. Featurebench: Benchmarking agentic coding for complex feature development. *Preprint*, arXiv:2602.10975.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024b. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

Wei Zhou, Xuanhe Zhou, Shaokun Han, Hongming Xu, Guoliang Li, Zhiyu Li, Feiyu Xiong, and Fan Wu. 2026c. Are we ready for an agent-native memory system? *Preprint*, arXiv:2606.24775.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025b. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. 2025. Where llm agents fail and how they can learn from failures. *Preprint*, arXiv:2509.25370.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2025. Agent-as-a-judge: Evaluate agents with agents. In *Forty-second International Conference on Machine Learning*.

A Survey Methodology and Coverage

A.1 Search Strategy and Eligibility

We froze the reviewed corpus on 20 July 2026. Discovery used Google Scholar query families combining *long-horizon*, *LLM agent*, *reliability*, *benchmark*, and domain terms such as *web*, *computer use*, *software engineering*, *tool use*, and *planning*. We complemented keyword search with backward and forward chaining from anchor benchmarks and methods already identified in each domain. Because the project did not preserve a complete historical log of every result seen before this revision, we do not report an artificial PRISMA-style initial-hit count.

Work was eligible when it evaluated an LLM-based agent in an interactive setting and when success depended on coupled actions, mutable or persistent state, delayed consequences, trajectory-level verification, or repeated-run reliability. We excluded static question answering, ordinary long-context processing, independent dialogue turns, and one-call tool demonstrations unless the work explicitly measured horizon-dependent behavior. We prioritized primary benchmark or method papers that disclosed tasks, environments, graders, or execution protocols; survey papers were used for orientation rather than as substitutes for primary evidence.

A.2 Evidence Extraction and Verification

For each benchmark, we extracted the task domain, scale, native extent unit, coupling mechanism, persistent state, correctness signal, resource or protocol budget, and source identity. New scholarly records were first located in Google Scholar and then checked against the official paper, proceedings page, arXiv record, or publisher metadata. This two-stage check reduces title collisions and prevents third-party descriptions from becoming bibliographic authority. Official product pages were used only for newly released commercial systems without a suitable first-party technical paper, and those systems are not used as comparable empirical evidence.

A.3 Coverage and Limitations

The resulting corpus covers web and browsing agents; computer, mobile, and embodied interaction; software engineering and terminal work; tool, API, app, and workplace workflows; planning and scientific tasks; model-level reasoning and learning; harness-level memory, verification, recovery, and oversight; and evaluation methodology. The complete benchmark inventory contains 64 entries in Appendix B. Coverage is intentionally evidence-weighted rather than exhaustive: fast-moving 2026 preprints, commercial systems with incomplete disclosures, and works without stable task or grader descriptions are discussed cautiously. Google Scholar is used as a discovery and cross-check layer, not as the sole authority for metadata or technical claims.

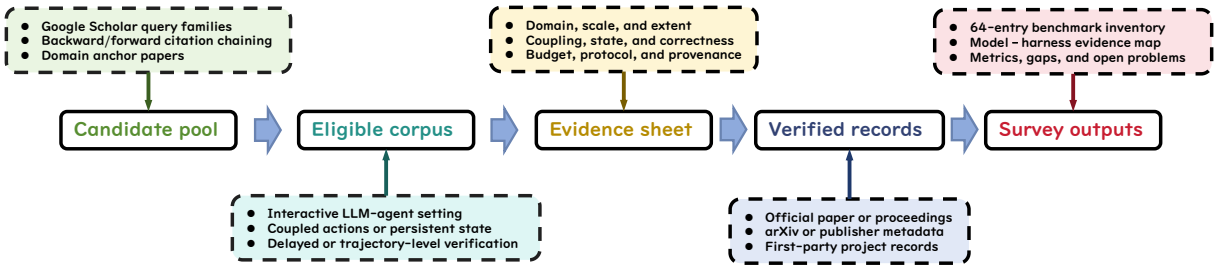

Figure 7: Survey methodology and evidence trail. Discovery, eligibility screening, structured extraction, and source verification lead to the evidence-coded synthesis used throughout the paper. The figure reports the final 64-entry benchmark inventory but deliberately does not invent unavailable historical screening counts.

B Benchmark Inventory and Evidence Sources

Table 7 lists all 64 benchmark entries considered in this survey and uses the same eight-field schema as the representative table in the main text. Completeness here refers to inventory coverage, not to the availability of an official task-length statistic. We assign a typed extent level only when an official source, or a separately identified external primary measurement, supports it; otherwise both length fields are marked NR (not reported). Prefix T denotes time-derived extent, S denotes native interaction counts, and B denotes protocol budgets or execution windows; numeric levels are comparable only within the same prefix and compatible measurement provenance, and budget codes additionally require their native unit. We prefer human/reference trajectories, then native execution counts, and finally official protocol budgets; rollout- or scaffold-specific values are retained only when explicitly labelled as such. Aggregate performance metrics are intentionally excluded because they belong to the evaluation protocol rather than the benchmark-inventory schema.

Table 7: Complete inventory of 64 long-horizon agent benchmark entries. Extent levels and per-task values follow the same evidence hierarchy and thresholds as Table 2; NR means that no supported official per-task extent was recorded, so no level is assigned.

Benchmark	Domain	Scale	Type	Per-task extent	Coupling	Persistent state	Correctness signal
Web navigation and browsing							
MiniWoB	Web UI	100 tasks	NR	NR	In compound tasks, earlier widget actions create page state required by later actions	Current page, widget values, and episode state	Environment reward after task completion
WebShop	Web Shopping	12,087 instructions	S2	11.3 states (mean; human expert); max 114	Product constraints couple search, comparison, option selection, and purchase	Visited products, selected options, cart, and session state	Task reward; exact success when reward equals 1
Mind2Web	Web Navigation	2,350 tasks	S1	7.3 actions (mean; reference demonstrations)	Cross-page actions depend on prior navigation and selections	Recorded DOM snapshots and prior actions in the demonstration trace (static dataset)	Reference element and action checks
WebLINX	Dialogue Web Navigation	2,337 demonstrations	S3	43 turns (mean; reference demonstrations)	Dialogue turns determine later browser actions	Recorded dialogue/action history, DOM snapshots, and screenshots (static demonstrations)	Reference intent, element, and text-action checks
WebArena	Web Navigation	812 tasks	T1	35.38 s (median; external 100-task sample)	Cross-page and cross-site subgoals	Website, account, and database state	Task-specific functional checks over final state or answer
WebCanvas	Live Web Navigation	542 tasks	S1	8.39 annotated steps/task (mean)	Page transitions and form edits create later prerequisites	Live page, DOM, form, and account state	Key-node URL, element-path, and value matches; step- and task-level checks
VisualWebArena	Visual Web Navigation	910 tasks	S1	9.6 steps (mean; official GPT-4V+SoM rollout; model/scaffold-specific)	Visual grounding across dependent website actions	Website, account, database, and task-relevant visual-content state	Functional final-state evaluators
WebVoyager	Open-Web Navigation	643 tasks	B2	15 steps (maximum; official evaluation budget)	Earlier web actions determine the page state available to later actions	Live browser session, navigation history, current page, and site state	Multimodal GPT-4V task-success judge over final answers and recent screenshots
Computer use, mobile, and embodied worlds							
OSWorld	Desktop Computer Use	369 tasks	T2	111.94 s (median; $n = 369$)	UI and file operations create later preconditions	Files, applications, documents, settings, and OS state	Execution-based final application, file, and OS checks
OSWorld 2.0	Professional Computer Workflows	108 tasks	T4	About 1.6 h (median; annotated/estimated); 69.6% $>$ 1 h	Interdependent cross-application workflows with dynamic updates	Workspace files, application/service records, user-profile data, messages, and injected updates	Fine-grained final-state checkpoints; functional checks with limited model-based judging
Windows Agent Arena	Windows Computer Use	154 tasks	S1	8.1 steps (mean; one-participant human evaluation)	Window, application, and file actions create later preconditions	Windows applications, files, settings, and desktop state	Task-specific final-state evaluators
OmniACT	Desktop/Web Control	9,802 tasks	S1	1 action (derived median; mean 1.66, range 1–11; official scripts)	In multi-action tasks, earlier script actions create the UI state required by later actions	N/A (static screenshot-to-action-sequence prediction)	Reference action-sequence and action-level checks
Android World	Mobile Computer Use	116 task templates	S1	6 optimal steps (median); range 1–60	Earlier UI actions mutate app/device state required by later actions	Android UI, app databases, records, files, and settings	Programmatic rewards over device and application state
B-MoCA	Mobile Device Control	131 tasks	S1	9 steps (derived median maximum episode length; range 4–$\geq$17)	Device-control actions depend on the changing screen state	Android UI, application state, language, and settings	Goal-state success checks
ALFRED	Embodied Household Tasks	8,055 demonstrations	S3	50 action steps (mean); 7.5 subgoals (mean)	PDDL-derived navigation and manipulation chains	Agent pose, object locations, receptacles, and object states	Task success and goal-condition success
BEHAVIOR-1K	Embodied Activities	1,000 activities	NR	NR	Object-state and spatial transitions are coupled by BDDL initial/goal predicates and physical feasibility	Scene graph, object poses, relations, and physical states	Goal predicates and task-success checks
RoboCerebra	Robotic Manipulation	60 test tasks / 600 rollouts	S5	2,972.4 simulation steps (mean; human trajectories)	Manipulation subgoals depend on completed prior operations	Scene, object states, robot state, and subtask progress	Simulator object-state predicates; exact plan matching and VideoQA checks

Table 7: Complete inventory of 64 long-horizon agent benchmark entries (continued).

Benchmark	Domain	Scale	Type	Per-task extent	Coupling	Persistent state	Correctness signal
Computer use, mobile, and embodied worlds (continued)							
WeaveBench	Hybrid Computer Use	114 tasks	S3	76 tool calls (median; best official Hybrid rollout/task); mean 88, range 14–471	GUI, CLI, code, and filesystem actions share artifacts	Desktop/application state, source/config files, artifacts, logs, and service status	Trajectory-aware agentic judge over artifacts, screenshots, logs, and actions, with shortcut detection
HeroBench	Virtual-World Planning	844 tasks (180 evaluated)	S3–5	Difficulty-stratum means 40–937 environmental actions (mean ± SD)	Resource dependencies and action order constrain later choices	Simulator location, inventory, equipment, skill levels, and resource counts	Goal-completion and partial-progress checks
Software engineering and terminal work							
SWE-bench	Software Engineering	2,294 tasks	T3	Estimated median bucket 15–60 min	Cross-file changes must jointly satisfy issue-specific and regression tests	Working tree, source, dependencies, build, and test state	FAIL_TO_PASS and PASS_TO_PASS tests
SWE-bench Verified	Software Engineering	500 tasks	T3	Estimated median bucket 15–60 min	Cross-file changes must jointly satisfy issue-specific and regression tests	Working tree, source, dependencies, build, and test state	FAIL_TO_PASS and PASS_TO_PASS tests
SWE-Cycle	Issue-Resolution Cycle	489 tasks	B4	90 min (isolated) / 3 h (FullCycle), maximum	Environment reconstruction, code implementation, and verification-test generation are cross-phase dependent	Repository, execution environment, reproduction, and test state	SWE-Judge static code review plus dynamic execution tests
SWE-bench-Live	Software Engineering	1,319 tasks (initial release)	B3–4	60 iterations (OpenHands) / 100 LLM calls (SWE-Agent), maximum	Cross-file changes must jointly satisfy fresh issue-specific and regression tests	Repository snapshot, issue context, working tree, and tests	Issue-specific executable tests
SWE-MERA	Software Engineering	728 samples (2025 snapshot)	NR	NR	Issue repair couples localization, edits, and validation	Repository snapshot, source files, dependencies, and tests	Repository-level executable tests
SWE-rebench V2	Software Engineering	32,079 tasks	NR	NR	Cross-language repair depends on build and test feedback	Repository, language toolchain, dependencies, and test state	Task-specific executable tests
SWE-Bench Pro	Professional Software Engineering	1,865 tasks	B3	50 turns (maximum; official evaluation budget)	Professional requirements couple cross-file implementation choices	Large repository, dependencies, build artifacts, and tests	Repository-level acceptance and regression tests
DeepSWE	Software Engineering	113 tasks	B4	2.5 h (maximum; official leaderboard budget)	Cross-file implementations must jointly satisfy task-specific behavior verifiers	Repository, source changes, build state, and test results	Hand-written program verifiers for task-specific acceptance
SWE-Marathon	Project-scale Software Engineering	20 tasks	T5	Expert-human estimate 40–400 h	Repeated project-scale implementation, build, and test loops	Task workspace, source/output artifacts, build/runtime state, and measured results	Tests, behavioral checks, judges, and performance gates
Feature Bench	Feature Development	200 tasks	B5	500 steps (maximum; default official evaluation budget)	Feature requirements couple architecture, implementation, and tests	Repository, feature artifacts, dependencies, and regression state	Feature-specific acceptance and regression tests
SWE-Explore	Repository Exploration	848 tasks	B1–3	8 search calls (LocAgent) / 25–30 steps (Mini-SWE-Agent/ AweAgent), maximum	Relevant code regions must be jointly selected and ranked under a fixed line budget	Repository snapshot, issue specification, and ranked code-region output (no code changes)	Ground-truth relevant-file and location labels
ReCUBE	Repository-Level Code Generation	366 masked-file instances	S2	20.3 turns (mean; GPT-5 Mini + CCE official rollout)	Cross-file symbols, dependencies, and documentation constrain masked-file reconstruction	Unmasked repository files, dependency specifications, documentation, and the reconstructed file	Usage-aware executable tests for internal logic and cross-file integration
SWE Atlas	Repository-Level Engineering	284 tasks	S3	56–58 tool calls (means; official mini-SWE-agent rollouts)	Repository context and runtime behavior jointly constrain Q&A, test-writing, or refactoring outputs	Repository, retrieved context, working tree, and test state	Programmatic checks plus task-specific rubric-based LLM judging

Table 7: Complete inventory of 64 long-horizon agent benchmark entries (continued).

Benchmark	Domain	Scale	Type	Per-task extent	Coupling	Persistent state	Correctness signal
Software engineering and terminal work (continued)							
Program Bench	Program Reconstruction	200 tasks	B5	6 h / 1,000 steps (maximum; official evaluation budget)	Observed behavior must be reconstructed across dependent modules	Generated repository, module interfaces, build, and execution state	Fuzzed end-to-end behavioral tests against reference executables
NL2Repo-Bench	Repository Generation	104 tasks	S3	86.0 tool calls (mean; Claude-Sonnet-4.5/OpenHands official rollout)	Natural-language requirements constrain interacting repository components	Generated files, interfaces, dependencies, build, and tests	Repository build and task-specific tests
RepoGenesis	Microservice Generation	30 eval / 76 train repositories	B1–4	5 iterations (MetaGPT) / 100 chat rounds (MS-Agent), maximum	README requirements couple service design, APIs, and implementation	Generated services, API schema, dependencies, and deployment artifacts	API and executable integration checks
SWE-EVO	Software Evolution	48 tasks	B4	100 iterations/calls (maximum; official evaluation budget)	High-level release requirements couple coordinated multi-file changes with regression preservation	Pre-release codebase, agent patch, dependencies, build state, and test results	Executable tests after each evolution stage
SWE-CI	Continuous Integration	100 tasks	B2	20 turns (maximum; official evaluation budget)	CI feedback drives successive maintenance decisions	Repository, workflow configuration, CI logs, and test state	CI jobs and executable test outcomes
SWE-Milestone	Continuous Evolution	98 graded milestones	T5	1–3 days (average human-estimated development time per milestone)	Milestone requirements and prior edits constrain later stages	Persistent repository, milestone artifacts, commits, and tests	FAIL_TO_PASS and PASS_TO_PASS tests at each milestone
Roadmap Bench	Version Upgrades	115 tasks	S4	134 agent steps (fleet mean across official evaluations)	A multi-target roadmap jointly constrains coordinated cross-file changes from source to target version	Repository versions, dependencies, migration state, and tests	Subtask-level executable tests against target-version behavior
SWE-Chain	Release-Level Upgrades	155 version transitions	S5	1,057–2,008 tool calls (per-chain means across official agents)	Chained package upgrades make each release depend on prior changes	Package repository, dependency versions, release state, and tests	Build and test checks after chained releases
ChainSWE	Multi-Bug Maintenance	304 bug instances (100 chains)	B3–4	30 min / 100 turns (maximum per bug instance)	Bug fixes interact through a shared evolving codebase	Persistent repository, prior fixes, issue state, and test results	Per-bug and final-chain executable tests
TEBench	Test Evolution	314 tasks	B3	30 min (maximum; official coding-agent runner timeout)	Stale, breaking, or missing tests depend on source evolution	Updated production code, evolving test suite, generated test patch, and test results	Developer-ground-truth affected-test labels; test executability, coverage-overlap, and patch-similarity checks
Terminal-Bench 2.0	Terminal / Systems	89 tasks	T4–5	Expert median bucket 1 h–1 day (74 timed tasks)	Commands, debugging, builds, and artifacts depend on prior state	Filesystem, packages, processes, artifacts, and logs	Task-specific executable tests in isolated containers
LongCLI-Bench	CLI Programming	20 tasks	T5	1,000+ min (mean; expert time)	Long command-line implementation sequences reuse prior artifacts	Filesystem, source, shell, processes, build state, and logs	Step-level and final executable checks
BuildBench	Software Build	148 test repositories	S1	6.6 error-resolution attempts (mean; official agent rollout)	Dependency and build commands create prerequisites for later stages	Source tree, dependencies, build artifacts, environment, and logs	Strict and flexible build-success checks
DI-BENCH	Dependency Inference	581 repositories	T3	20 min (maximum; official human study on 40 repositories)	Dependency choices constrain installation and repository execution	Manifests, lockfiles, repository, environment, and test state	Dependency matching, installation, and execution checks
CI-Repair-Bench	CI Repair	567 tasks	S2	28.4 LLM API calls/task (derived mean; GPT-5-mini official agent pipeline)	Code, workflow, environment, and dependency artifacts are jointly constrained by full CI re-execution	Repository, workflow files, CI logs, patches, and test state	CI workflow and test reruns
CLI-Tool-Bench	CLI Tool Generation	100 repositories	S3	48.68 steps (mean; best official Kimi-k2.5/Mini-SWE rollout)	Specification, implementation, packaging, and invocation are coupled	Generated repository, package metadata, build artifacts, and sandbox-visible file/system state	Black-box command-execution tests
SWE-InfraBench	Infrastructure Code	100 tasks	B1	1–2 LLM turns (official one-turn and two-turn solver configurations)	Infrastructure changes couple resources, configuration, and dependencies	AWS CDK repository, resource definitions, dependency graph, and test state	Provided tests over synthesized CloudFormation templates

Table 7: Complete inventory of 64 long-horizon agent benchmark entries (continued).

Benchmark	Domain	Scale	Type	Per-task extent	Coupling	Persistent state	Correctness signal
Software engineering and terminal work (continued)							
FrontierSWE	Frontier Technical Work	17 tasks	B5	20 h/task (protocol budget)	Open-ended technical requirements couple implementation with task-specific functional, research, or performance objectives	Task workspace, source/checkpoints, experimental variants, and build/benchmark results	Task-specific functional and performance checks; correctness and anti-cheat gates
PostTrain Bench	AI R&D / Post-training	28 task configurations	B5	10 h on one H100/ configuration (protocol budget)	Data preparation, training, evaluation, and tuning loops	Training data, scripts/configs, checkpoints, logs, and evaluation results	Held-out target-benchmark evaluation; LLM judge for contamination and model substitution
Tool, API, app, and workplace workflows							
ToolBench	Tool/API Use	1,200 test instructions	NR	NR	In multi-tool scenarios, later API calls depend on earlier outputs and argument choices	Call history, API outputs, selected tools, and intermediate results	LLM-based ToolEval task-success and pairwise-preference judgments over action sequences
BFCL	Function Calling	5,551 question–function–answer pairs	NR	NR	In multi-turn/multi-step subsets, later calls depend on prior outputs, user turns, and backend state	Backend API state, conversation history, available functions, and tool responses	Executable function-call and structured-match checks
AppWorld	App/API Workflows	750 tasks	S3	42.5 calls (mean; Test-N) / 46.8 (mean; Test-C)	Dependent API calls across multiple apps	Application databases, stateful execution-shell variables/results, and simulated date/time	State-based unit tests, including collateral-damage checks
WorkArena	Enterprise Web Workflows	19,912 instances	S2	13.7 oracle actions (derived weighted mean from official task-type statistics)	ServiceNow navigation and form/list operations are coupled by evolving page and record state	Browser session, ServiceNow records, forms, and task state	Task-specific final-record and interface-state evaluators
OfficeBench	Office Automation	300 tasks	B3	50 steps (maximum; official evaluation budget)	Cross-application document operations reuse intermediate artifacts	Filesystem, office documents, emails, calendar events, and application state	Exact-match, fuzzy-match, and execution-based task checks
Odyssey Bench	Office Application Workflows	602 tasks	S1–2	Majority require 3–15 execution turns	Long interaction histories constrain multi-step reasoning and actions across office applications	Multi-day dialogue history, filesystem, documents, emails, and calendar events	Exact, fuzzy, and execution-based checks over the saved final filesystem
TheAgent Company	Workplace Simulation	175 tasks	S2	27.2 steps (mean; best official Gemini-2.5-Pro/OpenHands rollout)	Browsing, coding, communication, and coworker interactions share subgoals	Organizational services, artifacts, messages, and coworker state	Deterministic or LLM-based result checks; subcheckpoint checks for partial credit
Constraint-heavy planning and scientific work							
Travel Planner	Travel Planning	1,225 queries	T3	About 12 min (mean; human annotation)	Itinerary choices share budget, temporal, route, and preference constraints	Static sandbox database, Notebook contents, constraints, and selected travel records	Rule-based checks over explicit and commonsense constraints
NATURAL PLAN	Natural-Language Planning	3,600 tasks	NR	NR	Temporal, availability, and ordering constraints jointly determine the final plan	N/A (one-shot planning from static tool outputs and constraints)	Parsed-plan exact match against the golden plan
Deep Planning	Travel/ Shopping Planning	360 cases	B5	400 tool calls (maximum; official evaluation budget)	Information gathering, local choices, and global constraints interact	Sandbox data, tool results, constraints, and partial plan	Code-based checks over parsed itinerary constraints; ground-truth cart-item matching
MinePlanner	Symbolic World Planning	45 tasks	S3	42 human-solution actions (derived median over 45 tasks); range 3–1,268	PDDL action preconditions and resource dependencies constrain ordering	Symbolic world, inventory, object relations, and goal state	Plan executability and goal-predicate satisfaction
Robotouille	Asynchronous Planning	300 tasks	S2–3	10–82 optimal steps (reported 20 single-agent task types)	Real delays and interruptions require interleaved dependent subplans	Objects, cooking state, timers, action queue, and subtask progress	Simulator goal-state and timing checks
ALE-Bench	Algorithm Engineering	40 full / 10 lite	B4	Contest-window median 4 h; agent cap 4 h/problem	Propose → implement → run → score/error feedback → refine	Current/best code, scores, feedback, and execution logs	Continuous problem score and normalized performance

C Domain-Term Word Cloud

Figure 8 provides a domain-term cloud derived from the titles of the 201 works cited in this survey. Closely related title variants are consolidated into normalized multiword concepts, and each concept is counted at most once per cited title. Font size is proportional to this title-level document frequency; colors are used only for visual separation and do not encode categories or importance.

Figure 8: Domain-term cloud of normalized multiword concepts in the titles of the 201 works cited in this survey. Font size is proportional to the number of cited titles matched to each concept.