Long-Horizon GUI Agents: A Survey of Benchmarks, Mitigation Methods, and Failure Types

Anonymous authors
Paper under double-blind review

Abstract

Graphical user interface (GUI) agents are autonomous systems that control computing devices through their native user interfaces. They perceive the screen, decide an action, and issue clicks, keystrokes, or touch gestures, without requiring the target software to expose an API. Driven by large multimodal models, these systems are moving from short research demonstrations to commercial deployments that run for hours and span dozens of applications. Evaluating their reliability at long task horizons is an open engineering problem. As the base models matured and commercial deployments grew, the evaluation community responded with a wave of new benchmarks and mitigation methods between 2024 and 2026, most of which appeared in the last year. We catalog twenty-five long-horizon GUI agent benchmarks across desktop, mobile, and web platforms and compare how each defines horizon, constructs tasks, and measures success. We catalog twenty mitigation methods and compare their mechanisms, claimed targets, and reported gains. We normalize the failure terminology used across these papers into a vocabulary of ten failure types, each described by at least two independent surveyed papers, and cross-analyze which types have dedicated methods. We find that horizon definitions are inconsistent across benchmarks, that most methods are evaluated on only one or two benchmarks, and that three failure types that recur widely in the benchmark literature (UI navigation knowledge gaps, catastrophic forgetting of task constraints, and misinterpretation of the task instruction) have few or no methods explicitly designed to fix them. A secondary analysis of published results shows that benchmark rankings of agents agree only weakly and that reported method gains decrease as benchmark horizon increases. The largest gains are reported on the shortest and most established benchmarks, which are also the ones that have been available long enough for methods to be tested on them. We argue that current evaluation practice is dominated by outcome-level metrics and underrepresents process-level reliability, which is the property that matters most at long horizons. We discuss the limitations of this analysis and list open problems for the field.

1 Introduction

Graphical user interface (GUI) agents are autonomous systems that control desktop computers, mobile devices, and web browsers through their native user interfaces. Unlike traditional software automation, which requires the target application to expose an API or scripting interface, a GUI agent interacts with software the way a human does: it perceives the screen through a screenshot or accessibility tree, decides an action, and issues a click, keystroke, scroll, or touch gesture. This design makes GUI agents application-agnostic. Any software a human can operate through a screen, a GUI agent can in principle operate too. The capability is enabled by large multimodal models that can read screens and reason about what to do next.

Commercial systems in this class now automate expense reporting, form filling, data migration, software testing, and enterprise workflows that previously required a human at a keyboard. As deployments mature, the tasks these systems handle have moved from short demonstrations of a few clicks to workflows that take dozens to hundreds of steps and span multiple applications and websites. A task that starts in a

web browser, pulls data into a spreadsheet, updates a database, and sends an email now runs as a single autonomous session.

This shift in task horizon creates a reliability problem that the field has not systematically characterized. Benchmarks built for short tasks do not exercise the failure modes that appear at long horizons, such as an agent forgetting a constraint it was given at step one, losing track of state when it switches from a browser to a spreadsheet, or misclicking once and derailing the remaining fifty steps. Between 2024 and 2026 a wave of new benchmarks has appeared specifically to evaluate agents at long horizons, and alongside them a wave of mitigation methods that try to make agents reliable on long tasks. Both waves are recent, fast-moving, and uncoordinated. Different benchmark papers define "long horizon" differently. Different method papers describe the failures they fix in different terms.

This paper catalogues both waves and connects them. We make four contributions.

First, we catalog twenty-five long-horizon GUI agent benchmarks published between 2024 and 2026. Each benchmark is compared on platform, horizon definition, task construction, and evaluation method. To our knowledge, no prior survey jointly catalogs long-horizon GUI benchmarks and maps mitigation methods to failure types.

Second, we catalog twenty mitigation methods that target long-horizon reliability. Each method is compared on mechanism, the failure type it claims to fix, the benchmarks it was tested on, and its reported gain.

Third, we normalize failure terminology. Benchmark and method papers describe failures using inconsistent terms. We extracted every failure description from the surveyed papers, grouped synonyms, and produced a shared vocabulary of ten failure types. A failure type is included if at least two independent surveyed papers describe it. Each type carries a reference list of the papers that describe it.

Fourth, we cross-analyze which failure types have dedicated mitigation methods. The distribution is uneven. Planning and Grounding errors each have six or seven dedicated methods. Three other failure types have one or zero: Navigation (finding where a function lives in an unfamiliar application), Forgetting (ignoring a constraint stated earlier), and Instruction (misreading the task).

The remainder of the paper is organized as follows. Section 2 positions this survey against prior work. Section 3 provides background on GUI agent architecture, foundation models, and the short-horizon benchmarks that preceded the work surveyed here. Section 4 describes the methodology: how candidate papers were found, the inclusion criteria, and the extraction process. Section 5 presents the benchmark catalog. Section 6 presents the failure vocabulary. Section 7 presents the mitigation method catalog. Section 8 presents the cross-analysis. Section 9 presents a secondary analysis of published results. Section 10 discusses limitations. Section 11 lists open problems. Section 12 concludes.

2 Related Surveys

Several recent surveys cover GUI agents or LLM agent evaluation. Each has a different organizing axis from this paper.

Shi et al. (2026b) survey trust in GUI agents, organized by pipeline stage (perception, reasoning, interaction). Long-horizon behavior is one subsection. Zhang et al. (2025) survey LLM-based GUI agents by framework, data, action model, and benchmark, positioned as a practical cookbook for building agents rather than an analysis of failure coverage. Sager et al. (2026) survey agents for computer use by architecture, covering 87 agents and 33 datasets. A survey of GUI agent benchmarks (Chen et al., 2026c) organizes by task type rather than by horizon. A survey of agent memory mechanisms (Hu et al., 2025) covers memory forms, functions, and dynamics across all agent types, contrasting memory with context engineering, but is not GUI-specific. Mohammadi et al. (2025) survey evaluation and benchmarking of LLM agents generally, organized by evaluation objective rather than by horizon or failure type. Hu et al. (2026) survey reinforcement learning for GUI agents, organized by training paradigm. Qi et al. (2026), appearing shortly after our literature cutoff, survey trustworthy agentic AI across safety, robustness, privacy, and security, but the scope is general agentic systems rather than GUI agents and the organizing axis is threat category rather than horizon.

Wang et al. (2026) propose a seven-category failure taxonomy for long-horizon agentic systems across domains including tool use, web, and OS. Their taxonomy is cross-domain and not GUI-specific, and their paper is a diagnostic benchmark rather than a survey. Several of their categories appear in the vocabulary presented in Section 6 because they recur across the GUI papers we surveyed.

This paper differs from all of the above in three ways. It is scoped to long-horizon GUI tasks, it covers benchmarks and mitigation methods in the same catalog, and it maps each mitigation to the failure types it targets.

3 Background

3.1 GUI Agent Architecture

A GUI agent operates a computer the way a person does. It looks at the screen, decides what to do, and issues an input action. Perception usually means a screenshot, which works on any platform. Some agents also read the browser DOM or the operating system accessibility tree when those exist. A language model handles the decision step. It receives the observation, the task instruction, and its prior actions, then picks the next move. The action step translates that choice into raw input: a click at pixel coordinates, a typed string, a key combination, or a scroll. This observe, think, act loop repeats until the task completes or the agent gives up.

This architecture differs from tool calling agents in one fundamental way. A tool calling agent invokes functions that a developer exposed on purpose. Each function has a name, a typed signature, and a documented contract. If the developer did not write a function for some capability, the agent cannot use it. A GUI agent has no such dependency. It operates the interface that already exists for human users, so any application a person can run, the agent can in principle run too. It requires no API, no plugin, and no integration work. That independence is what makes the approach application agnostic. The same agent that fills out a spreadsheet can also install a VPN client or edit a video, and it needs no code change to do so.

This generality comes with a cost. The agent perceives pixels rather than semantics, which makes it fragile. A button that shifts three pixels is still the same button to a person, but the agent has to rediscover it. A modal dialog that appears unexpectedly blocks every later action until the agent notices and dismisses it. Each pass through the observe, think, act loop is a point of failure, and those failures compound over the length of a task. Task horizon is therefore the central axis along which GUI agent research has developed.

3.2 Foundation Models for GUI Interaction

Early GUI agents were built by prompting generalist multimodal models. GPT-4V (OpenAI., 2023) was the first widely used option, and GPT-4o, Gemini (DeepMind., 2025), and the open Qwen-VL series (Bai et al., 2025) followed. These models were trained for broad visual question answering rather than screen operation. They can describe what a screenshot shows and reason about what to do, but they cannot reliably turn that reasoning into a precise action. Generalist models fail in the gap between "click the Save icon in the toolbar" and the pixel coordinate (842, 63).

This gap is called grounding, the problem of mapping a semantic target to a location on screen, and it is the core challenge that specialist GUI models were built to solve. SeeClick (Cheng et al., 2024) trained on GUI grounding pairs and showed that a small model with the right supervision can outperform a large model without it. CogAgent (Hong et al., 2024) added a high resolution visual branch so the model could resolve small interface elements. Ferret-UI (You et al., 2024) targeted mobile screens. OS-Atlas (Wu et al., 2025) scaled grounding data across desktop, mobile, and web. Aguvis (Xu et al., 2024) and OpenCUA (Wang et al., 2025) added full interaction trajectories to the training mix, moving beyond grounding-only supervision. UI-TARS (Qin et al., 2025) pushed this line to its current endpoint with a single model trained end to end to observe, reason, and act.

The field has shifted from prompting to training as a result. Prompting a generalist model is cheap to iterate on but plateaus quickly, because the model was never asked to operate a screen during training. Training a

specialist costs more and needs interaction data at scale, but it produces an agent that grounds reliably and has learned how to sequence actions through an interface. Most competitive systems in 2025 use a specialist model for perception and action, sometimes with a generalist model in a planner role above it.

3.3 The Evolution from Short to Long Horizons

Benchmark design has consistently lagged behind agent capability, so the history of GUI benchmarks has been one of catching up. The LLM agent era began in 2023, and Mind2Web (Deng et al., 2023) collected roughly 2,350 tasks as static traces across 137 real websites. The tasks were realistic, but the evaluation was offline. An agent was scored on whether its step matched the recorded human action. This approach penalizes valid alternative paths and cannot measure recovery from error.

The next generation of benchmarks ran agents in live environments. WebArena (Zhou et al., 2024) hosted working replicas of an online store, a forum, a code hosting site, and a content management system across 812 tasks, and it scored agents on whether the final site state was correct. AitW (Rawles et al., 2023) released 715 thousand mobile interaction episodes in 2023, and AndroidWorld (Rawles et al., 2024) followed with 116 tasks executed on a live Android emulator. OSWorld (Xie et al., 2024) brought the same approach to a full operating system, defining 369 tasks on a real Ubuntu desktop with scripted checks on the final system state. These benchmarks set the template the field still uses: real software, live execution, and programmatic verification. Several benchmarks bridged the gap between these eras. WebVoyager (He et al., 2024) ran agents on live commercial websites rather than self-hosted replicas. VisualWebArena (Koh et al., 2024) extended WebArena with vision-dependent tasks. ScreenSpot (Cheng et al., 2024) isolated the grounding sub-problem as its own benchmark, and ScreenSpot-Pro (Li et al., 2025b) later extended it to high-resolution professional software. OmniACT (Kapoor et al., 2024) combined desktop and web tasks in one harness. Mobile-Agent (Wang et al., 2024) and UFO (Zhang et al., 2024) contributed early multi-agent frameworks that decompose tasks across planner and executor roles.

These benchmarks also exposed their own limits. At its 2024 release, the best agent completed about 12 percent of OSWorld tasks. Within eighteen months, specialist models and hybrid frameworks pushed past 60 percent (Song et al., 2025a), and by 2026 frontier commercial agents approached the 72 percent human baseline. As the established benchmarks moved toward saturation, the field needed evaluations that remained hard. The success metric on these benchmarks is also binary, which means a run that fails on step 2 and a run that fails on step 19 both score zero. The second agent therefore gets no credit for doing nearly all the work. Most tasks also stay inside one application, which does not match how people actually use computers. People move data between a browser, a spreadsheet, and an email client over dozens of steps, during which they lose track, backtrack, and recover. The 2025 and 2026 benchmark wave this survey covers was built to measure exactly that kind of work, using longer horizons, more applications per task, instructions drawn from real users, and metrics that separate where an agent failed from whether it failed.

4 Methodology

4.1 Search Strategy

Candidate papers were found by searching arXiv, Semantic Scholar, and Google Scholar between March and May 2026 for combinations of the following terms: GUI agent, computer use agent, web agent, mobile agent, long horizon, multi-step, benchmark, evaluation, and failure. Reference lists of found papers were traced forward and backward. The arXiv listing pages for cs.CL, cs.AI, and cs.HC were monitored weekly through the cutoff date. The literature cutoff is April 2026.

4.2 Inclusion Criteria

A paper is in scope if it meets one of two conditions.

First, it introduces a benchmark for GUI agents where tasks require at least fifteen interaction steps. We use fifteen interaction steps as a working threshold, while noting that some benchmarks define horizon using alternative units such as sub-goals, tool calls, or sub-workflow counts. Benchmarks that report horizon in

other units are included if the paper characterizes its tasks as long-horizon or multi-step, and the reported unit is noted in the benchmark catalog.

Second, it proposes a mitigation technique for long-horizon execution and evaluates that technique on a benchmark that meets the first condition.

GUI means interaction via pixels, screenshots, accessibility trees, or document object models on desktop, mobile, or web. Pure API tool-calling agents, command-line-only agents, single-step grounding work, and embodied robotics are out of scope.

4.3 Selection Outcome

The search returned 78 candidate papers. After full reading, 41 are in scope: 25 benchmarks and 20 mitigation methods. Some papers contribute both a benchmark and a method and are counted in both catalogs. The 37 excluded papers were ruled out because they are not GUI (for example, pure tool-calling or command-line agents), not long-horizon (single-step grounding benchmarks), or not agent-focused (general LLM evaluation). Figure 1 shows the selection process.

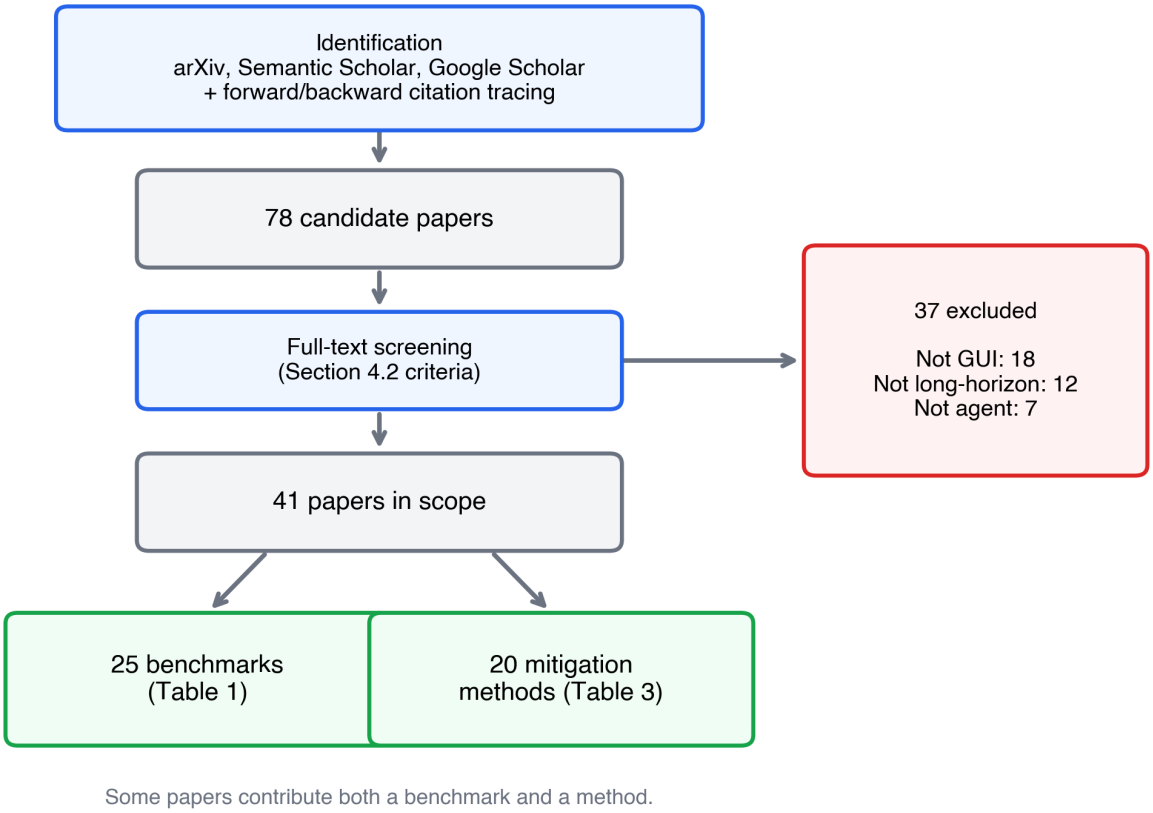

Figure 1: Paper selection process.

4.4 Extraction

For each in-scope paper, the following fields were extracted by reading the abstract, introduction, method, and evaluation sections: platform, number of tasks, horizon definition and value, task construction method, evaluation method, key challenge, key metric, and failure descriptions. Extraction was done by reading each

paper. Fields that a paper does not specify are marked not stated in the catalog tables. For the method papers, an additional pass assigned a verdict of primary, secondary, or not addressed for each failure type. The rule was the following. Primary means the paper's abstract or introduction frames the method as a solution to that failure type. Secondary means the paper mentions the failure as a benefit or side effect but it is not the main motivation. Not addressed means the paper does not discuss the failure type. These verdicts were assigned by a single coder. While inter-rater reliability was not measured, all coding decisions were based on explicit definitions and supported by recorded evidence, with the supporting quotes recorded in a coding table available as supplementary material. This is noted as a limitation in Section 10.

5 Benchmark Landscape

Twenty-five long-horizon GUI agent benchmarks are compared in Table 1. Benchmarks are grouped by platform. Figure 2 shows publication dates by quarter. Twenty-two of the twenty-five benchmarks were published in 2025 Q2 or later. The three earlier ones are WindowsAgentArena (Bonatti et al., 2024), ST-WebAgentBench (Levy et al., 2024), and A3 (Chai et al., 2025). This surge followed two enabling developments: the release of multimodal models capable of reliable screen understanding, and the first commercial GUI agent products whose deployment failures created demand for harder, longer benchmarks. The benchmark community responded quickly, which is why the catalog below is dominated by work from a single year.

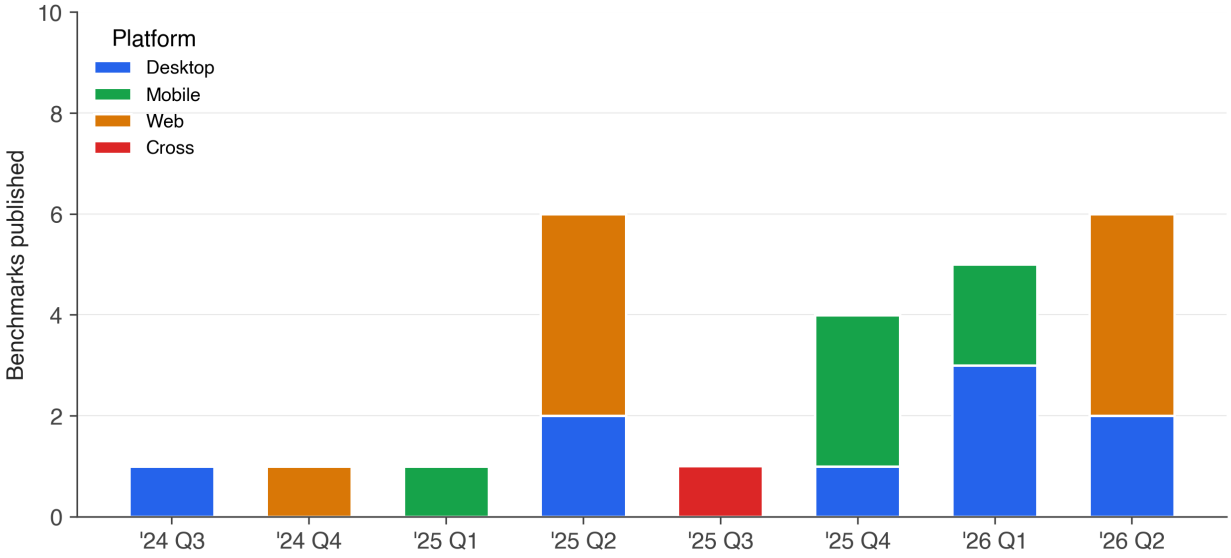

Figure 2: Long-horizon GUI benchmark publications by quarter.

Table 1: Long-horizon GUI agent benchmarks.

Benchmark	Platform	Horizon	Key challenge	Key metric
NaturalGAIA (Zheng et al., 2025)	cross-platform	Causal Pathways of 1-2, 3-4, 5-7 atomic task nodes	276 tasks as causally dependent atomic step chains across platforms where one error collapses the pathway	Weighted Pathway Success Rate (WPSR), best model 34.6%
CarePilot (Ghosh et al., 2026)	desktop	8-24 steps	1,100 healthcare GUI tasks spanning 8-24 steps on DICOM/EHR/LIS; actor-critic agent with dual memory and grounding tools	task accuracy 51.45% vs GPT-5 36.19% on CareFlow

Benchmark	Platform	Horizon	Key challenge	Key metric
Gym-Anything / CUA-World (Aggarwal et al., 2026)	desktop	CUA-World-Test 200-step budget; CUA-World-Long 500-step budget, tasks often need 200+ steps for human	12,000 tasks over 200 professional applications weighted by economic contribution; framework turns any software into a testable environment	pass rate via checklist VLM rubric; Gemini 3 Flash 7.5% on CUA-World-Long under 500-step budget (GPT-5.4 reaches 27.5% only under extended 2000-step budget)
OS-HARM (Kuntz et al., 2025)	desktop	15 steps default	150 OSWorld safety tasks spanning deliberate misuse, prompt injection, and benign tasks inviting costly mistakes	unsafe action rate via LLM judge (0.76/0.79 F1 vs human)
OS-Marathon (Wu et al., 2026a)	desktop	evaluated at 50/100/150/200 step budgets	242 repetitive tasks (expense reports, transcripts) where agents must repeat identical sub-workflow across up to ~30 data items	Sub-Workflow Accuracy (SWA), fraction of iterations correctly completed
OSWorld-Human (Abhyankar et al., 2025)	desktop	human gold averages 4.6-13.6 single-action steps, 3.2-8.8 grouped steps per app	human gold trajectories for 369 OSWorld tasks; stresses step efficiency, top agents use 1.4–2.7× extra steps	Weighted Efficiency Score (success weighted by human/agent step ratio)
OSWorld-MCP (Jia et al., 2025)	desktop	15 steps and 50 steps	158 MCP tools across 7 apps; agent must choose tool call vs GUI action each step	task accuracy, tool invocation rate (TIR), average completion steps
ProSoftArena (Ai et al., 2026)	desktop	Overall avg 12.9 steps, range 2-109. By level: L1 avg 5.1	436 tasks in 13 professional tools (SolidWorks, ArcGIS, ChemDraw) needing domain expertise; five graded capability tiers	success rate per capability tier; best agent 24.4% L2, 0% L3
WindowsAgentArena (Bonatti et al., 2024)	desktop	Paper characterizes comparable benchmarks as about 10-20 steps	154 multistep tasks in real Windows OS across 11 programs (office, browser, IDE, system, media)	task success rate (device state check); 19.5% agent vs 74.5% human
WindowsWorld (Li et al., 2026b)	desktop	Avg min expert steps: L1=9.67, L2=18.13, L3=27.81	professional cross-app Windows workflows from 16 occupations; 78% of 181 tasks span multiple applications	intermediate check score (avg 4.97 sub-goal checkpoints per task)
A3 (Chai et al., 2025)	mobile	Easy N<7, Medium 7<=N<=11, Hard N>11	100 tasks across 20 live online proprietary Android apps where internal state cannot be instrumented	Essential State Achieved Rate (ESAR) via MLLM judge
AndroTMem (Shi et al., 2026a)	mobile	avg 32.1, max 65	1,069 Android tasks avg 32.1 steps; later steps depend on sparse intermediate state from earlier ones	Task Completion Rate; ASM lifts TCR 5% to 30.16% over baselines
AndroidLens (Cao et al., 2025)	mobile	26.1 avg steps	571 mobile GUI tasks averaging 26 steps across 74 Chinese and English apps with real anomalies preserved	Average Task Progress (milestone completion); best agent 12.7% SR, 50.47% ATP
ColorBench (Song et al., 2025b)	mobile	avg 13.13+ steps	175 long horizon mobile tasks averaging 13+ steps; graph supports multiple valid paths and backtracking	success rate, completion rate, atomic task capability
MobileBench-OL (Wu et al., 2026b)	mobile	Long-Horizon subset requires >=20 steps, avg 22.73 steps	1080 tasks on 80 Chinese apps; subsets stress ≥20 step memory, UI exploration, injected popups/delays	Success Rate plus Sub-condition Success Rate via XPath rule checks
MobileWorld (Kong et al., 2025)	mobile	avg 27.8 completion steps	201 Android tasks averaging 27.8 steps, 62.2% span apps; adds clarification dialogs and MCP calls	task success rate, deterministic check via backend database inspection

Benchmark	Platform	Horizon	Key challenge	Key metric
Online-Mind2Web (Xue et al., 2025)	web	easy $\leq$5, medium 6-10, hard $\geq$11 human reference steps	300 tasks on 136 live websites; exposes over-optimism in prior benchmarks by showing frontier agents drop 20 to 60 points from self-reported WebVoyager scores under rigorous human evaluation	human-evaluated success rate; best agent Operator 61%; WebJudge auto-eval reaches 85% agreement with human
HealthAdminBench (Bedi et al., 2026)	web	many tasks involve 15 or more subtasks	135 healthcare admin tasks across simulated EHR, payer portals, fax; stresses document handling and data transfer between systems	task success and subtask success (deterministic + LLM judge, 1,698 subtasks)
Mind2Web 2 (Gou et al., 2025)	web	>50 actions	130 long horizon live web search tasks with time varying answers requiring cited synthesis across many sites	partial completion and success rate via rubric tree judge agent
Odysseys (Jang et al., 2026)	web	100-step budget cap	200 live web tasks from real user browsing history spanning multiple sites; avg 6.1 graded rubrics each	rubric score (averaged, perfect) and Trajectory Efficiency (score/step)
RealWebAssist (Ye et al., 2025b)	web	17.6 steps per task average	1,885 sequential spoken instructions from real users; ambiguous references depend on prior steps and user routines	task success rate; best model 14.0% vs human 93.4%
RiskWebWorld (Chen et al., 2026b)	web	20 step cap per episode	1,513 ecommerce risk investigation tasks across 8 domains; uncooperative live sites with CAPTCHAs and popups	task success rate (exact match or LLM judge); best 49.1%
ST-WebAgentBench (Levy et al., 2024)	web	step count not reported; avg ~4 min per task; avg 4.2 to 18.6 policies per task across subsets; paper describes tasks as long-horizon	375 enterprise web tasks carrying 3,057 safety policies the agent must obey while completing work	Completion Under Policy (CuP) and per dimension Risk Ratio
WebChoreArena (Miyai et al., 2025)	web	step count not reported; horizon characterized by task tags such as Long-Term Memory (multi-page recall) and Massive Memory (large single-page extraction)	532 tedious WebArena tasks stressing massive memory, arithmetic over observations, and recall across multiple pages	task accuracy; GPT-4o 6.8%, Gemini 2.5 Pro 44.9%
WebForge (Yuan et al., 2026)	web	50-action upper bound per task	934 auto-generated self-contained websites with injected popups, cookie dialogs, latency; 7-axis difficulty control across 7 domains	task success via final-state output match (answer or operation code)

5.1 Desktop Benchmarks

Desktop benchmarks run the agent inside a full operating system instead of a browser tab. The agent inherits the full machine state: installed applications, a real filesystem, window management, and workflows that cut across several programs. This setting makes the long horizon problem concrete, since a task that reads as one sentence in the instruction can unfold into dozens of clicks spread over three different applications. Four of the nine benchmarks in this group extend OSWorld (Xie et al., 2024) directly, reusing its tasks or its virtual machine harness: OS-HARM (Kuntz et al., 2025), OSWorld-Human (Abhyankar et al., 2025), OSWorld-MCP (Jia et al., 2025), and WindowsWorld (Li et al., 2026b). That shared lineage gives the OSWorld branch dense method coverage, while the rest of the group is newer and far more thinly reported.

The OSWorld derivatives have shifted from measuring whether a task gets done to measuring how well it gets done. OSWorld-Human (Abhyankar et al., 2025) pairs 369 OSWorld tasks with human gold trajectories and reports a Weighted Efficiency Score. Agents that succeed still take 1.4 to 2.7 times more steps than a person, so a plain success rate hides real inefficiency. OS-Marathon (Wu et al., 2026a) stretches 242 repetitive tasks across 50, 100, 150, and 200 step budgets and scores partial credit with Sub-Workflow Accuracy, which shows where an agent loses track of the task as the run grows longer. OSWorld-MCP (Jia et al., 2025) equips the agent with 158 MCP tools alongside the GUI on 361 tasks. At every step the agent must choose between a direct tool call and raw clicking, which makes tool selection itself the capability being evaluated. OS-HARM (Kuntz et al., 2025) builds 150 safety tasks on the OSWorld environment, mixing deliberate misuse, prompt injection, and benign tasks prone to costly mistakes. Taken together, these four benchmarks show that completion rate is no longer the most informative number on OSWorld. Efficiency, endurance, tool choice, and safety are the dimensions where agents still differ.

Professional software is the other axis, and it is where scores fall sharply. ProSoftArena (Ai et al., 2026) covers 436 tasks in 13 expert tools such as SolidWorks, ArcGIS, and ChemDraw, stratified into five capability tiers. The best agent reaches 24.4% at level two and 0% at level three, which is the sharpest ceiling reported anywhere in this group. CareFlow (Ghosh et al., 2026) narrows the domain to healthcare with 1,100 tasks over DICOM viewers, EHR systems, and LIS software at 8 to 24 steps each. The underlying assumption is that clinical workflows need their own benchmark rather than a slice of a general one. The Windows pair makes the same argument from the operating system side. WindowsAgentArena (Bonatti et al., 2024) reports 19.5% agent success against 74.5% for humans on 154 multistep tasks. WindowsWorld (Li et al., 2026b) pushes further into occupational realism with 181 workflows drawn from 16 professions. Most of them, 78%, span more than one application, and subgoal checkpoints allow partial progress to appear in the score rather than being lost to a binary pass or fail outcome. The gap between a general agent and a trained human widens as the software moves from simple consumer applications toward tools that professionals are trained to use.

Gym-Anything, also called CUA-World (Aggarwal et al., 2026), sits at the far end of the horizon axis. It offers roughly 12,000 tasks across 200 professional applications with a 200-step evaluation budget on tasks that often require more than 200 steps for a human, and its Long split allows over 500 steps, roughly an order of magnitude past most of the group. That scale is both the benchmark's appeal and its main limitation. Method coverage concentrates on the OSWorld lineage because the harness is shared and results are directly comparable, so the benchmarks that push hardest on horizon length and domain depth have the thinnest leaderboards. The pattern across the desktop group is consistent: the harder a benchmark makes the task resemble real professional work, the fewer agents have been evaluated on it, and the worse those agents do. Reading only the well populated benchmarks therefore gives an optimistic picture of where desktop agents actually stand.

5.2 Mobile Benchmarks

Mobile benchmarks share a verification problem that desktop benchmarks mostly avoid. Android apps expose almost no internal state to the evaluator, so there is no filesystem to diff and no document to reopen. A task that runs across twenty screens of a live banking app leaves nothing behind to inspect. The six benchmarks in this group are best read as six different answers to that scoring problem. The device itself adds a second constraint, because phone screens are small, scrolling is constant, and the intermediate state an agent needs to carry forward is sparse and easy to lose between screens.

Three of the six build the difficulty directly into task structure. AndroTMem (Shi et al., 2026a) has the longest mobile horizon in this survey at 32.1 average steps with a maximum of 65, and it constructs its 1,069 tasks so that later steps depend on sparse intermediate state produced earlier. That design makes memory rather than grounding the bottleneck. MobileBench-OL (Wu et al., 2026b) works at larger scale with 1,080 tasks across 80 Chinese apps. Its Long-Horizon subset requires at least 20 steps and averages 22.73, while other subsets deliberately inject popups, latency, and forced UI exploration. ColorBench (Song et al., 2025b) is shorter at roughly 13 steps but compensates through task structure. It represents each of its 175 tasks as a graph that permits multiple valid paths and explicit backtracking, so the benchmark can reward an agent that recovers from a wrong turn instead of penalizing it.

The remaining three split along the realism axis. A3 (Chai et al., 2025) runs its 100 tasks on 20 live proprietary Android apps whose internal state cannot be instrumented at all, which forces it to score an Essential State Achieved Rate from screenshots with an MLLM judge. AndroidLens (Cao et al., 2025) goes further and deliberately keeps the real anomalies it finds across its 74 Chinese and English apps rather than scrubbing them. The best agent reaches only 12.7% success rate there against 50.47% Average Task Progress, a gap of nearly 4x that shows how much a binary metric hides at this horizon. MobileWorld (Kong et al., 2025) makes the opposite design choice. It controls its 201 tasks end to end and verifies completion by inspecting a backend database, which gives it a deterministic check that the other mobile benchmarks in this group do not have. It also adds clarification dialogs and MCP tool calls, and 62.2% of its tasks span apps. The cost is giving up the live noise that A3 and AndroidLens keep. Neither side of this tradeoff is clearly better, and the group as a whole relies heavily on partial credit metrics because success rate alone collapses toward zero past 20 steps.

Horizon and method coverage do not correlate across this group. AndroTMem is the longest at 32.1 average steps, followed by MobileWorld at 27.8 and AndroidLens at 26.1, with ColorBench the shortest at about 13. Yet only two of the six have an external mitigation method reported on them. ClawGUI (Tang et al., 2026) reaches 17.1% on MobileWorld's GUI-only split and NaturalGAIA (Zheng et al., 2025) reports results on A3. AndroTMem is evaluated only by its own ASM module, and AndroidLens, ColorBench, and MobileBench-OL have no method results at all. Method developers still default to AndroidWorld (Rawles et al., 2024), which is not in this catalog because its horizon is short. The mobile benchmarks that are hardest and most recent are therefore the ones with the least evidence about what actually helps.

5.3 Web Benchmarks

The web group has converged on a single premise: agents score better on sandboxes than on the real internet, and the gap is large enough to change conclusions. Online-Mind2Web (Xue et al., 2025) makes this concrete by rerunning 300 tasks across 136 live websites and finding that agents lose 23 to 59 points relative to the scores their authors reported on WebVoyager (He et al., 2024). The paper pairs that finding with WebJudge, an automatic evaluator that reaches 85% agreement with human raters, because a live benchmark's credibility depends on the reliability of its judge. Mind2Web 2 (Gou et al., 2025) extends the same idea over time. Its 130 live search tasks run past 50 actions and have answers that drift week to week, so the agent must synthesize and cite sources rather than match a cached string. Taken together, these two papers argue that both the environment and the evaluator must be live for the numbers to be trustworthy.

A second cluster grounds tasks in observed user behavior rather than in tasks imagined by benchmark authors. Odysseys (Jang et al., 2026) mines 200 tasks from genuine browsing histories and gives agents a budget of 100 steps. Grading runs against an average of 6.1 rubrics per task, which acknowledges that real goals rarely have a single binary endpoint. RealWebAssist (Ye et al., 2025b) goes further in addressing ambiguity. It replays 1,885 sequential spoken instructions across tasks that average 17.6 evaluated steps each, where a phrase like "the second one" only resolves if the agent tracked what came before. The best agent manages 14.0% against a human baseline of 93.4%. That is the widest human gap in this group, and it came from underspecified language rather than from step count, which suggests instruction grounding is a harder ceiling than planning depth on the open web.

Most of the remaining entries focus on a single failure mode. WebChoreArena (Miyai et al., 2025) builds 532 new tedious tasks on WebArena's environment that force recall across many pages and long tables. GPT-4o scores 6.8%, which points to memory rather than reasoning as the bottleneck for tedious work. HealthAdminBench (Bedi et al., 2026) decomposes 135 healthcare administration workflows into 1,698 subtasks spanning EHR, payer, and fax systems, and it is one of the few web benchmarks where the domain itself is resistant to automation by design. RiskWebWorld (Chen et al., 2026b) runs 1,513 ecommerce risk investigations under a cap of 20 steps on live sites that present CAPTCHAs and other uncooperative behavior, so the environment actively interferes with the agent. ST-WebAgentBench (Levy et al., 2024) attaches 3,057 safety policies to 375 enterprise tasks and reports Completion Under Policy, which penalizes an agent for finishing a task while breaking a rule. WebForge (Yuan et al., 2026) takes the opposite design choice from nearly every other benchmark in this group. It generates 934 synthetic websites of its own and injects popups, cookie walls, and latency along seven difficulty axes, which gains control at the cost of realism.

The tension running through this group is between realism and reproducibility, and no benchmark has resolved it. Live benchmarks like Odysseys, RiskWebWorld, and Mind2Web 2 are faithful but drift, so a score reported this year cannot be replicated next year. WebForge is perfectly reproducible but synthetic, and ST-WebAgentBench sits in between on hosted enterprise stacks. Horizon length is also uneven across the group. Odysseys and Mind2Web 2 push past 50 to 100 actions, while RiskWebWorld caps at 20 and WebForge stays closer to a conventional episode. Method coverage tends to follow benchmark age. Online-Mind2Web has the broadest set of agents evaluated under one judge, which gives its inflation finding more weight. Most of the 2025 entries report results from only a handful of systems, so cross benchmark comparison remains thin. The consistent signal is that whenever a web benchmark adds real users, latency, or policy constraints, frontier agent performance drops sharply, and the cause is different each time.

5.4 Cross-Cutting Observations

Multi-application tasks are a common design trend. Most benchmarks in the catalog include tasks that span multiple applications or websites. Two benchmarks report the fraction explicitly: WindowsWorld (Li et al., 2026b) reports 78 percent of its 181 tasks spanning multiple applications and MobileWorld (Kong et al., 2025) reports 62 percent of its 201 tasks spanning apps. Others describe multi-application or cross-system workflows without a count, including CarePilot (Ghosh et al., 2026), HealthAdminBench (Bedi et al., 2026), OSWorld-MCP (Jia et al., 2025), Gym-Anything (Aggarwal et al., 2026), and NaturalGAIA (Zheng et al., 2025). The field is converging on the view that coordinating across applications is central to long-horizon difficulty.

Horizon definitions are inconsistent. Most benchmarks measure horizon in steps, but the counted unit varies. Some papers count agent actions, some count human reference steps, some count milestones or sub-goal checkpoints, and some evaluate at multiple fixed step budgets. Two benchmarks, ST-WebAgentBench (Levy et al., 2024) and WebChoreArena (Miyai et al., 2025), do not report a step count at all and characterize horizon qualitatively. This inconsistency makes cross-benchmark comparison difficult and is a barrier to progress.

Success rates collapse at long horizons. Frontier agents now approach the human baseline on the established short-horizon benchmarks discussed in Section 3.3, but the best-agent scores reported in Table 1, as of each benchmark's publication, fall sharply as tasks get longer and more realistic. CUA-World-Long (Aggarwal et al., 2026) reports a 7.5 percent pass rate under its standard 500-step budget for the best frontier model, and 6.0 percent for a model that scores above 70 percent on OSWorld-Verified. ProSoftArena (Ai et al., 2026) reports 0 percent at its hardest capability tier. OS-Marathon (Wu et al., 2026a) reports 0 percent end-to-end success on its Expense Report domain across every agent and step budget. AndroidLens (Cao et al., 2025) reports 12.7 percent on tasks averaging 26 steps. RealWebAssist (Ye et al., 2025b) reports 14.0 percent against 93.4 percent for humans. All five of these benchmarks were published between December 2025 and April 2026 and tested frontier models from that period. These numbers frame the problem this survey addresses: on realistic long-horizon tasks, current agents fail most of the time, and a binary success rate near zero tells a method developer nothing about what to fix. This collapse suggests that current benchmarks operate in a regime where failure analysis, rather than success measurement, is the primary source of insight.

Few benchmarks diagnose why an agent failed. Most benchmarks report a binary success or failure per task, which cannot distinguish an agent that misclicked on step two from one that forgot a constraint on step forty. A minority report partial credit, such as OS-Marathon's Sub-Workflow Accuracy (Wu et al., 2026a), AndroidLens's Average Task Progress (Cao et al., 2025), and WindowsWorld's sub-goal checkpoints (Li et al., 2026b). Partial credit shows where progress stopped but not why. Fewer still classify failures into categories: NaturalGAIA (Zheng et al., 2025) attributes failure to a specific atomic step in its causal pathway, and OS-HARM (Kuntz et al., 2025) labels unsafe actions by type. For benchmarks that report outcome only, the data needed to prioritize one failure type over another is not available from the published result.

5.5 Limitations of Current Evaluation Paradigms

The observations in Section 5.4 point to a deeper measurement issue. Almost all surveyed benchmarks evaluate at the outcome level. They check whether the final environment state matches a target, then report success or failure. This is appropriate for short tasks where one decision determines the outcome. For long-horizon tasks it loses most of the available signal.

Long-horizon reliability is a process-level property. An agent that fails on step 3 of 50 and an agent that fails on step 48 of 50 receive the same score. The first agent could not even start. The second nearly completed the task. They likely failed for different reasons and need different fixes. An outcome metric cannot distinguish them. The same is true for error propagation: an Accumulation failure is by definition a sequence of dependent errors, but a single end-state check sees only the last one.

A few surveyed benchmarks offer process-level alternatives. Sub-Workflow Accuracy in OS-Marathon (Wu et al., 2026a) credits each completed iteration. Average Task Progress in AndroidLens (Cao et al., 2025) credits each milestone. WindowsWorld (Li et al., 2026b) scores intermediate sub-goal checkpoints. These give more information than binary success. They still do not say which failure type stopped progress, only where progress stopped. NaturalGAIA (Zheng et al., 2025) goes further and attributes failure to a specific atomic step.

The conclusion is not that success rate is wrong. Success rate is necessary because deployment requires task completion. It is insufficient because at the success rates the field currently achieves, the difference between two agents is not how often they succeed but how and where they fail. Evaluation that reports only outcome cannot guide work on the failure types catalogued in the next section. Process-level evaluation that locates and classifies failure is a precondition for making use of the failure vocabulary. Gonzalez-Pumariega et al. (2026) make a complementary point empirically. They run agents on the same OSWorld task many times and find that the same agent succeeds on some runs and fails on others, which they attribute to execution stochasticity, task ambiguity, and behavioral variability. Outcome-level success rates average over this variance and hide it.

6 Failure Vocabulary

On several surveyed benchmarks, the best reported agent solves fewer than one task in five: AndroidLens (Cao et al., 2025) at 12.7 percent, RealWebAssist (Ye et al., 2025b) at 14.0 percent, ClawGUI on MobileWorld (Tang et al., 2026) at 17.1 percent, and WindowsAgentArena (Bonatti et al., 2024) at 19.5 percent. The remaining tasks are all marked as failures, but failures are not all alike. An agent that forgot a constraint, an agent that misclicked, and an agent that misunderstood the task need three different fixes. A shared vocabulary for describing failure types is the prerequisite for knowing which fix to build. This section builds that vocabulary.

We normalize terminology used across papers into a shared vocabulary. It was built by extracting failure descriptions from all in-scope papers and grouping synonyms. A failure type is included if it is described by at least two independent surveyed papers. The References column in Table 2 lists the surveyed papers that describe that failure type.

The Label column gives the short name used in the rest of this paper.

Planning. The agent understood the task but decomposed it into the wrong subgoals or ordered the steps incorrectly. Erdogan et al. (2025) quantify the cost on WebArena-Lite: a base prompted executor with no planner solves 9.85% of tasks, while the full system with a separate planner and finetuned executor reaches 57.58%. Wang et al. (2026) call this Planning Error, NaturalGAIA (Zheng et al., 2025) reports Planning and Reasoning Errors, and WebDART (Yang et al., 2025c) frames it as a failure to split a web chore into navigation, extraction, and execution phases. The boundary with Instruction lies in the starting point, since a Planning failure begins from a correct reading of the task and goes wrong when the agent turns that reading into steps.

Table 2: Failure vocabulary for long-horizon GUI agents. Names in the References column refer to entries in Tables 1 and 3, where full citations are given. HORIZON refers to the diagnostic taxonomy of Wang et al. discussed in Section 2.

Label	**Full name**	**What it means**	**References**
Planning	Planning Error	The plan or a sub-plan is wrong, or steps are in the wrong order.	HORIZON, NaturalGAIA, OS-Marathon, MobileUse, RegionFocus, CoAct-1, PLAN-AND-ACT, WebDART
Accumulation	History Error Accumulation	An early mistake compounds through later steps and derails the task.	HORIZON, NaturalGAIA, OS-Marathon, UltraCUA, GTA1, VeriGUI, CoAct-1
Memory	Memory Limitation	The trajectory exceeds the context limit or key information is truncated.	HORIZON, OS-Marathon, AgentProg, AndroTMem, Mind2Web 2, OCR-Memory
Forgetting	Catastrophic Forgetting	A stated constraint or fact is present in context but ignored later.	HORIZON, MobileBench-OL, AgentProg, AndroTMem, ST-WebAgentBench, WebChoreArena
Environment	Environment Error	The agent acts on stale state or fails to notice that the UI changed.	HORIZON, MobileBench-OL, AgentProg, VeriGUI, AndroidLens, ProSoftArena, RiskWebWorld
Instruction	Instruction Error	The agent misunderstands or only partly understands the task description.	HORIZON, MobileBench-OL, Online-Mind2Web, AndroidLens, CoAct-1, OS-HARM, ProSoftArena
Assumption	False Assumption	The agent assumes something about the environment that is not true.	HORIZON, AgentProg, VeriGUI, AndroTMem, WebChoreArena, WindowsWorld
Grounding	Perception and Grounding Error	Wrong click coordinates, misidentified UI element, or a hallucinated element that does not exist.	ColorBench, MobileBench-OL, NaturalGAIA, OS-Marathon, MobileUse, UltraCUA, GTA1, VeriGUI, RegionFocus, UGround, V-Droid
Navigation	UI Navigation Knowledge Gap	The agent does not know where a function lives in the application or does not know the application's interaction conventions.	ColorBench, MobileBench-OL, NaturalGAIA, MobileUse, GTA1, VeriGUI, RegionFocus, CarePilot, CUA-Skill, Mobile-Agent-v3
CrossApp	Cross-App State Coordination	The agent loses or corrupts state when switching between applications.	ColorBench, NaturalGAIA, OS-Marathon, MobileWorld, AgentProg, AndroTMem, WindowsWorld

Accumulation. One wrong action early in a trajectory does not stay local, because it puts the agent on an unexpected screen and every later decision inherits that offset. CoAct-1 (Song et al., 2025a) shows on OSWorld that failure rate rises with step count, and routing click chains through a code interpreter brings average steps on solved tasks to 10.15 versus 15.22 for a pure GUI agent. NaturalGAIA (Zheng et al., 2025) constructs tasks so that a single step error triggers deterministic Path Collapse, and Wang et al. (2026) label the category History Error Accumulation. Accumulation is a downstream effect rather than a root cause. The first mistake is usually a Planning, Grounding, or Environment failure, and Accumulation names what that mistake does to the rest of the run.

Memory. The trajectory grows past what the model can hold, so information physically leaves the context window. OCR-Memory (Li et al., 2026a) reports that its approach cuts text tokens injected per step from 3,980 to 596 relative to a text-retrieval baseline by rendering the log as compressed images. AndroTMem (Shi et al., 2026a) builds tasks averaging 32.1 steps and up to 65 and finds that degradation with horizon length is driven by memory failures rather than by perception or local action errors. Wang et al. (2026) call this Memory Limitation. The contrast with Forgetting is that in a Memory failure the information is no longer in the prompt at all, so no amount of attention can recover it.

Forgetting. The constraint or fact is still in the prompt, but the agent stops acting on it. Levy et al. (2024) report that completion under policy falls from 18.2% with one active policy to 7.1% with five or more, even though every policy was given up front. AgentProg (Tian et al., 2025) names this Failure to Remember Critical Information, Wang et al. (2026) call it Catastrophic Forgetting, and WebChoreArena (Miyai et al., 2025) builds 247 of its 532 tasks around retaining facts across later steps (its Long-Term Memory category). Unlike a Memory failure, nothing was truncated, so the agent still has the information but does not use it. Forgetting may grow more common as context windows expand, since a larger window keeps the constraint in the prompt but also keeps more distractors alongside it. This trade-off is discussed in Section 11.

Environment. An Environment failure occurs when the screen changes while the agent is deliberating and the agent does not notice the shift, so it acts on stale state. RiskWebWorld (Chen et al., 2026b) injects CAPTCHAs, popups, and dynamic content shifts and finds specialized GUI models collapse to 0% success when these hijackings appear. Wang et al. (2026) call it an Environment error, AgentProg (Tian et al., 2025) calls it Poor Understanding of Dynamic Environments, and VeriGUI (Zhang et al., 2026) counters it by predicting the expected screen change per step and checking the prediction against the actual screen. An Assumption failure invents a state that never existed, while an Environment failure misses a real change that occurred.

Instruction. An Instruction failure happens before the first action, when the agent misreads or only partly reads the task description, so even a flawless execution completes the wrong task. Xue et al. (2025) attribute 57.7% of their best agent's failures on Online-Mind2Web to filter and sorting errors, a class where the agent searches the site but drops a condition the instruction stated. Wang et al. (2026) name the category Instruction, and several benchmark papers describe it as ambiguity or clarification failure. Planning failures start from a correct reading and go wrong at step ordering, while Instruction failures start from a wrong reading and execute a different task than the one asked.

Assumption. An Assumption failure, which Wang et al. (2026) call a False Assumption, occurs when the agent acts on an unverified belief about the environment and treats something as true that it never observed. AgentProg (Tian et al., 2025) attacks this with a Global Belief State module that tracks what has actually been observed under partial observability, and removing it drops AndroidWorld success from 78.0% to 53.9%. VeriGUI (Zhang et al., 2026) makes the check explicit by predicting the expected screen change per step and verifying the prediction against the real screen. An Environment failure misses a change that really happened, while an Assumption failure invents a state that never existed.

Grounding. Grounding is the failure of mapping a decision to pixel coordinates, where the agent knows which element it wants and clicks the wrong place, misreads which widget is which, or targets an element that is not on the screen at all. Twelve surveyed papers name it, the most of any type, and it has no analog in tool calling agents. UGround (Gou et al., 2024) trains a grounding model on 10 million synthetic web elements and lifts ScreenSpot accuracy from 53.4% to 73.3%, while RegionFocus (Luo et al., 2025) zooms into subregions after a click error to recover the target. NaturalGAIA (Zheng et al., 2025) groups the same

failure under Perceptual Error. The distinction from Navigation is a matter of knowledge, since a Grounding failure knows where the button is and still misses it, while a Navigation failure does not know where the button is in the first place.

Navigation. A Navigation failure means the agent knows what to do but cannot find where to do it, because it does not know where a function lives in the application or does not understand the application's conventions. CUA-Skill (Chen et al., 2026a) responds by hand building a library of 478 atomic skills across 17 Windows applications, and the skill layer alone adds 15.6 points to a GPT-5 baseline on WindowsAgentArena. NaturalGAIA (Zheng et al., 2025) calls this Knowledge Deficit, and MobileUse (Li et al., 2025a) both lists Navigation as a failure category and adds a Proactive Exploration module that crawls the app before acting. Navigation is therefore not a clicking problem, since a Grounding failure aims at a known target and misses, while a Navigation failure never finds the target to aim at.

CrossApp. A CrossApp failure happens when state is lost or corrupted during the switch between applications. WindowsWorld (Li et al., 2026b) isolates the effect with a step matched comparison across task levels, where final success drops from 46.15% to 14.29% at equal horizon length, and the authors attribute the drop to application switching rather than to step count. AndroTMem (Shi et al., 2026a) builds 1,069 cross app tasks averaging 32.1 steps and proposes an anchored state memory that retrieves intermediate results by reference when the agent returns from another app. OS-Marathon (Wu et al., 2026a) and ColorBench (Song et al., 2025b) report the same pattern on desktop and mobile. CrossApp overlaps with Memory because the app switch is when facts get dropped, but the cause differs. A Memory failure runs out of context budget, while a CrossApp failure can happen with context to spare because the agent did not carry the right facts across the boundary.

Three of the ten types are specific to GUI interaction and do not arise in tool-calling or API-based agents. Grounding is the failure of mapping a decision to a location on screen, which only exists when actions are clicks and taps rather than named function calls. Navigation is the failure of knowing where a function lives in an application, which only exists when the agent must find a capability by exploring an interface rather than reading an API specification. CrossApp is the failure of carrying state across application boundaries, which only exists in environments with multiple applications and no shared data layer. The other seven types appear across agent domains and are described by both GUI and non-GUI papers in the surveyed literature.

While the vocabulary distinguishes failure types, the surveyed literature rarely distinguishes their severity. A Grounding miss that clicks the wrong button is usually low impact. The agent observes the wrong screen, recognizes the error, and retries. A CrossApp failure that pastes the wrong customer record into a database, or a Forgetting failure that ignores a "do not delete" constraint, can corrupt state in a way no later step can undo. Both count as one failure in a binary success metric. They differ greatly in real-world consequence. Two surveyed benchmarks engage with severity: OS-HARM (Kuntz et al., 2025) labels actions by safety category, and ST-WebAgentBench (Levy et al., 2024) scores policy violations separately from task failure. The other benchmarks treat all failures as equivalent. Connecting failure type to failure severity, and especially to whether a failure is recoverable, is an open direction discussed in Section 11.

7 Mitigation Method Landscape

Section 6 catalogued the ways an agent can fail. This section catalogues the methods that try to prevent those failures. Reading the twenty method papers, a pattern emerges: each method intervenes at one point in the agent's observe, think, act loop, and methods that intervene at the same point share a design. We group methods by that intervention point, which produces five categories.

Context Management methods change what the agent carries between steps. A long task produces more history than the model can usefully hold, so these methods compress, restructure, or index the history rather than feed it in raw. Hierarchical methods change how the agent decides. They decompose the task into subtasks, add reflection layers between the planner and the executor, or assign roles to multiple model instances. Grounding methods change how the agent acts, by improving the mapping from a semantic decision such as "click the Save button" to a pixel coordinate. Hybrid and Verifier methods wrap the act step.

They either replace fragile click sequences with programmatic actions or check the action before and after it runs. Skill Library and Training methods change what the agent knows, by storing reusable procedures the agent can retrieve or by training the model end to end on interaction data so the knowledge lives in the weights.

The category a method belongs to tells the reader which part of the loop it protects, which constrains which of the Section 6 failure types it can address. A Context Management method cannot fix a Grounding error, because the error happens after the history is read. A Grounding method cannot fix a Forgetting error, because the agent already decided the wrong action before it tried to ground it. Section 8 formalizes this mapping.

Table 3 shows the catalog, grouped by mechanism. The Mechanism column shows which category each method belongs to, and Sections 7.1 through 7.5 discuss each category in the same order. The Primary targets column lists failure types the method's abstract or introduction frames as the problem it solves. The Secondary targets column lists failure types the paper mentions as a benefit or side effect. Both the failure type verdicts and the mechanism assignment were made by a single coder, as noted in Section 4.4.

Table 3: Long-horizon GUI agent mitigation methods.

Method	**Mechanism**	**What it does**	**Primary targets**	**Secondary targets**	**Tested on**	**Key result**
AgentProg (Tian et al., 2025)	Context Management	reframes interaction history as a program; variables persist while finished loop iterations and untaken branches get pruned	Planning, Memory, Environment, CrossApp	Forgetting, Assumption	AndroidWorld, AW-Extend	78.0% vs 73.3% prior SOTA on AndroidWorld
AndroTMem (Shi et al., 2026a)	Context Management	anchored state memory that indexes intermediate results so later steps can retrieve them by reference	Memory, Forgetting, CrossApp	—	AndroTMem-Bench	ASM lifts TCR 5% to 30.16% over baselines
CarePilot (Ghosh et al., 2026)	Context Management	actor-critic agent with dual memory and grounding tools for healthcare GUI tasks	Grounding, Memory	Accumulation, Navigation	CareFlow, OpenHospital OOD	task accuracy 51.45% vs GPT-5 36.19% on CareFlow
OCR-Memory (Li et al., 2026a)	Context Management	stores agent history as indexed images, retrieves by selecting visual anchor indexes then fetching verbatim text	Memory	—	Mind2Web, AppWorld, RULER NIAH	58.1% vs 56.2% ACON on AppWorld avg SR
Mobile-Agent-v3 (Ye et al., 2025a)	Hierarchical	multiple GUI-Owl instances act as specialized role agents sharing observations and reasoning traces	Planning, Grounding	Memory, Environment, Navigation, CrossApp	OSWorld-Verified, AndroidWorld, AndroidControl, ScreenSpot-V2	73.3 AndroidWorld, 37.7 OSWorld-Verified
MobileUse (Li et al., 2025a)	Hierarchical	three-tier reflection (action, trajectory, global) triggered on demand by confidence, plus proactive app exploration	Accumulation, Assumption	Planning, Environment, Grounding, Navigation	AndroidWorld, AndroidLab	62.9% vs 59.5% prior SOTA on AndroidWorld

Method	Mechanism	What it does	Primary targets	Secondary targets	Tested on	Key result
NaturalGAIA (Zheng et al., 2025)	Hierarchical	hierarchical agent with task parser and workflow manager for causal-pathway tasks	Planning, CrossApp	Accumulation, Memory, Assumption, Grounding, Navigation	NaturalGAIA, A3	Weighted Pathway Success Rate, best model 34.6%
OS-Marathon (Wu et al., 2026a)	Hierarchical	sub-workflow decomposition with templated repetition for long repetitive tasks	Planning	Memory, Instruction	OS-Marathon	Sub-Workflow Accuracy
PLAN-AND-ACT (Erdogan et al., 2025)	Hierarchical	separate finetuned planner and executor models with dynamic replanning, trained on synthetic plan data	Planning	Memory, Environment, Forgetting	WebArena-Lite, WebArena, WebVoyager	57.58% vs 49.1% prior SOTA on WebArena-Lite
WebDART (Yang et al., 2025c)	Hierarchical	splits web tasks into navigation, extraction, and execution subtasks; replans when new page shortcuts appear	Planning	Accumulation, Forgetting	WebChoreArena, WebArena	31.2% vs 22.4% AWM baseline on WebChore-Arena (GPT-5)
GTA1 (Yang et al., 2025a)	Grounding	samples multiple action proposals per step, judge model picks best, plus RL-trained click grounding model	Planning, Grounding	Accumulation	ScreenSpot-Pro, ScreenSpot-V2, OSWorld-G, OSWorld	45.2% vs 42.9% CUA o3 on OSWorld
RegionFocus (Luo et al., 2025)	Grounding	after a click error, zooms into subregions, predicts an action per crop, tracks tried spots visually	Grounding	Accumulation, Forgetting	ScreenSpot-Pro, WebVoyager	61.6% vs 47.8% baseline on ScreenSpot-Pro with Qwen2.5-VL-72B
UGround (Gou et al., 2024)	Grounding	trains a grounding model on 10M synthetic web elements to map element descriptions to pixel coordinates	Grounding	—	ScreenSpot, Multimodal-Mind2Web, AndroidCon-trol, OmniACT	73.3% vs 53.4% SeeClick on ScreenSpot grounding accuracy
CoAct-1 (Song et al., 2025a)	Hybrid/Verifier	orchestrator delegates subtasks to GUI operator or programmer agent writing code to bypass brittle click sequences	Accumulation, Grounding	Planning, Memory	OSWorld, Win-dowsAgentArena	60.8% vs 55.98% (Agent S2.5) on OSWorld
UltraCUA (Yang et al., 2025b)	Hybrid/Verifier	agent mixes GUI primitives with programmatic tool calls to shorten fragile action chains	Accumulation	Grounding, Memory, CrossApp	OSWorld-Verified, Win-dowsAgentArena	41.0% vs 33.3% OpenCUA-32B base on OSWorld

Method	Mechanism	What it does	Primary targets	Secondary targets	Tested on	Key result
V-Droid (Dai et al., 2025)	Hybrid/Verifier	uses LLM as verifier to score extracted candidate UI actions rather than generate actions directly	—	Grounding, Planning	AndroidWorld, AndroidLab, MobileAgent-Bench	59.5% vs 54.3% prior SOTA on AndroidWorld; 0.7s per decision
VeriGUI (Zhang et al., 2026)	Hybrid/Verifier	predicts expected screen change per step, verifies against actual screen, retries with corrected action on mismatch	Accumulation, Environment, Assumption	Grounding	AndroidControl-High, AITW-Gen, GUI Odyssey, MiniWoB++	52.5% Recovery Success Rate vs 45.5% UI-TARS-7B
CUA-Skill (Chen et al., 2026a)	Skill/Training	library of reusable parameterized Windows GUI skills with execution graphs; agent retrieves and composes them per step	Navigation	Planning, Accumulation, Memory, Grounding	WindowsAgentArena, self-synthesized tasks	57.5% best-of-3 on WindowsAgentArena; +15.6pt over no-skill GPT-5
ClawGUI (Tang et al., 2026)	Skill/Training	open source framework: online RL with dense process rewards, standardized evaluation, real device deployment	—	Memory	MobileWorld GUI-Only	17.1% vs 11.1% MAI-UI-2B on MobileWorld; 95.8% reproduction rate
Gym-Anything / CUA-World (Aggarwal et al., 2026)	Skill/Training	framework turns any software into a testable environment with auto-generated rubrics	—	Memory, Forgetting	CUA-World-Long	Gemini 3 Flash 7.5% pass rate on CUA-World-Long (500-step budget)

The subsections below discuss each mechanism category in the order shown in Table 3. Within each category, the methods share a core design choice but differ on how to implement it.

7.1 Context Management

The four methods in this group (AgentProg (Tian et al., 2025), AndroTMem (Shi et al., 2026a), CarePilot (Ghosh et al., 2026), OCR-Memory (Li et al., 2026a)) address the mismatch between how fast a GUI trajectory grows and how much of it the model can usefully hold. A long task produces dozens of screenshots and action records. Feeding all of them into the context drowns the signal, while truncating naively causes the agent to forget a value it fetched twenty steps earlier. The failures this group targets are memory decay, loss of intermediate results, and cross app handoffs where data pulled from one application has to survive long enough to be entered into another.

All four methods compress or restructure history in some way, which reflects a shared design premise that raw history cannot be carried forward intact. They differ on what the compressed form should be. AgentProg (Tian et al., 2025) treats the interaction log as a program. Finished loop iterations are pruned while variables the remaining code still references are kept. This gives the agent a principled rule for what to forget, and it lifts AndroidWorld success from the prior 73.3% state of the art to 78.0%. The risk is that the program abstraction has to be inferred online, so a step misread as a completed loop body discards state the task still needs. OCR-Memory (Li et al., 2026a) takes the opposite approach and refuses to summarize at all. History is stored as indexed images, the agent selects a segment index from the anchored image, and then the original text is fetched verbatim from the stored log. The name refers to Optical Context Retrieval

rather than character recognition. The underlying claim is that summarization loses procedural detail and free-form generation hallucinates, so the method keeps the exact text and retrieves it only when needed. It reaches 58.1% on AppWorld. The cost is a heavier retrieval path and a dependence on querying the anchor index correctly.

AndroTMem (Shi et al., 2026a) and CarePilot (Ghosh et al., 2026) fall between these two approaches. AndroTMem writes intermediate results into an anchored state memory for later lookup. ASM improves task completion rate by 5 to 30.16 percentage points depending on the base agent. The absolute number stays low, which indicates that anchoring helps but memory is not the only failure source on that benchmark. CarePilot pairs dual memory with an actor critic loop and grounding tools for clinical software. It reaches 51.45% where GPT-5 alone scores 36.19%. That gap is large, but it comes from one specialized domain and the memory component is ablated in the paper's own experiments, which show both short-term and long-term memory contribute independently. The larger problem for the whole group is measurement, since each method evaluates on its own benchmark set with no overlap across the group. Because none of the benchmarks overlap, the published numbers do not support cross method comparison. No paper reports retrieval precision, so we do not know how often AgentProg prunes something it still needed, how often OCR-Memory fetches the wrong frame, or how often AndroTMem's anchors go stale once the app navigates away from the indexed screen. Until recall error rates appear next to task success, choosing between these compression strategies is not supported by evidence.

7.2 Hierarchical Decomposition and Reflection

The six methods in this group are Mobile-Agent-v3 (Ye et al., 2025a), MobileUse (Li et al., 2025a), NaturalGAIA (Zheng et al., 2025), OS-Marathon (Wu et al., 2026a), PLAN-AND-ACT (Erdogan et al., 2025), and WebDART (Yang et al., 2025c).

Hierarchical methods address long horizon failure by avoiding reasoning about the whole task at once. They split the problem across levels so that no single model call carries the full planning burden. Most papers in this group name Planning as their primary target, which fits the mechanism. If a flat agent plans badly because the horizon is too long, the fix is to shorten the horizon each component sees. PLAN-AND-ACT (Erdogan et al., 2025) is the most direct implementation of this idea. It trains a planner and an executor as separate finetuned models. When execution drifts, the planner regenerates the plan. The split lifts WebArena-Lite from 49.1% to 57.58%, an 8.5 point gain that combines the separation of planner and executor with finetuning on synthetic plan data. WebDART (Yang et al., 2025c) applies the same split inside a single web task, breaking work into navigation, extraction, and execution subtasks. When a page reveals a shortcut, it replans around the original decomposition. On WebChoreArena with GPT-5 it reaches 31.2% against a 22.4% prior best. Both approaches pay the same cost: the planner cannot see the raw screen state the executor sees, so it can issue subgoals the executor cannot reach. Detecting and repairing that mismatch requires extra model calls.

Two other methods specialize the decomposition for a particular task shape. OS-Marathon (Wu et al., 2026a) targets long repetitive workflows. It cuts a task into sub-workflows and reuses a template across repeated instances instead of replanning each one from scratch. The assumption is that most long tasks are long because they repeat rather than because they branch. NaturalGAIA (Zheng et al., 2025) starts from the opposite assumption. Its LightManus agent pairs a task parser with a workflow manager for causal pathway tasks, where one step produces evidence that a later step consumes, often across applications. The hierarchy exists to carry dependencies forward rather than to amortize repetition. Even so, the best model reaches only 34.6% Weighted Pathway Success Rate, which says more about the difficulty of causal pathway tasks than about the method. Both systems report mainly on their own benchmarks, which makes their gains difficult to compare against the planner and executor split that PLAN-AND-ACT and WebDART evaluate on shared testbeds.

Mobile-Agent-v3 (Ye et al., 2025a) and MobileUse (Li et al., 2025a) add reflection on top of the decomposition. Mobile-Agent-v3 runs multiple GUI-Owl instances as specialized role agents that share observations and reasoning traces. It reaches 73.3 on AndroidWorld. The decomposition here is by role rather than by subtask. The shared trace lets a later agent catch an earlier agent's mistake. MobileUse keeps a single

agent but reflects at three tiers covering the action, the trajectory, and the global plan. A confidence signal gates the reflector at the action level, so the agent only pays that reflection cost when it is uncertain. The trajectory and global reflectors fire on a fixed schedule. The agent also explores the app proactively before committing to a plan. The gain is smaller at 62.9% over a 59.5% prior AndroidWorld result, but it is the only method in this group whose primary targets are Accumulation and Assumption rather than Planning. This difference marks the main dividing line in the group. Decomposition methods assume the agent fails because it plans badly, while MobileUse assumes it fails because small errors compound unnoticed. The two claims are not mutually exclusive, but no paper here tests both interventions on the same base model to separate the effects. Most also do not report on benchmarks that push past a few dozen steps, so whether any of these hierarchies scales to tasks with hundreds of actions remains untested.

7.3 Grounding Improvement

The three methods in this group are GTA1 (Yang et al., 2025a), RegionFocus (Luo et al., 2025), and UGround (Gou et al., 2024).

Grounding errors happen when the agent has already decided what to do but picks the wrong pixel. The plan itself is correct, but the click lands on the wrong element or on empty space beside it, and the trajectory drifts. Over a long horizon these errors compound. One misclick can open the wrong menu, toggle a setting, or leave the interface in a state the policy no longer recognizes. The methods in this group all treat coordinate prediction as the real bottleneck and address it directly rather than through better planning or longer memory.

UGround (Gou et al., 2024) fixes grounding at training time. The authors build a 10 million element corpus, about 90 percent synthetic web screenshots, and train a dedicated grounding model on it. The assumption is that scale closes the gap, and on ScreenSpot it does: UGround reaches 73.3% element accuracy where SeeClick sits at 53.4%. The cost is paid once before deployment, so inference stays cheap. The limitation is coverage, because the corpus consists of web screenshots, and desktop applications or dense professional tools fall outside that distribution, so the margin is not guaranteed to carry over. RegionFocus (Luo et al., 2025) takes the opposite route and fixes grounding at inference time, reactively. After a click misses, it crops the screen into subregions, predicts a fresh action inside each crop, and overlays visual markers on locations already tried so the model does not repeat itself. This lifts ScreenSpot-Pro from 47.8% to 61.6% with no retraining. The trade-off is latency, since every retry requires another model call, and the benefit only appears on screens dense enough that a single forward pass tends to miss.

GTA1 (Yang et al., 2025a) sits between the two approaches. It samples several candidate actions per step and has a judge model rank them before anything touches the screen, which makes the search proactive rather than reactive. It also trains its grounding with RL on click accuracy, a training time fix like UGround but far smaller in scale. On OSWorld, GTA1 with an o3 planner reaches 45.2%, ahead of OpenAI's CUA o3 at 42.9%, which is a modest gain. UGround also reports task success on Mind2Web-Live and AndroidWorld, and RegionFocus reports on WebVoyager, but all three report their largest gains on grounding benchmarks, and a 20 point jump on ScreenSpot does not translate one for one into finished tasks. All three methods isolate grounding as a separable subproblem, which is a shared design choice rather than an explicit claim in any of the papers. They differ on whether to pay for the fix with data, with retries, or with per step sampling. We did not find any of the three reporting the downstream cost of the misclicks that still get through. A grounding model at 73% accuracy still misses roughly one click in four, which on a task of fifteen steps means several expected misses per episode. We did not find these papers measuring how many of those residual misses are recoverable versus how many mutate state in a way the agent cannot undo.

7.4 Hybrid Action and Verification

The four methods in this group are CoAct-1 (Song et al., 2025a), UltraCUA (Yang et al., 2025b), V-Droid (Dai et al., 2025), and VeriGUI (Zhang et al., 2026).

The methods in this group share a common premise: individual GUI actions fail at a nontrivial rate, and over a long horizon that per step error compounds into near certain task failure. Two responses to this

problem appear in the literature. Hybrid action routes around the fragile steps by replacing clicks with code, while verification keeps the click chain but checks each step before the error propagates. CoAct-1 (Song et al., 2025a) applies the routing approach at the subtask level. An orchestrator assigns each subtask either to a GUI operator or to a programmer agent that writes and executes code. On OSWorld, CoAct-1 reaches 60.8%, which is ahead of the prior best Agent S2.5 at 55.98%. The paper's own ablation shows that hybrid routing adds about 10 points over a GUI-only variant. The gain comes from work such as bulk file edits, which is deterministic in code but brittle through the GUI. The trade-off is that routing introduces a new decision point, and a wrong routing choice wastes the turn. The code path also exposes failure modes that a misclick cannot reach. UltraCUA (Yang et al., 2025b) applies the same substitution at the action level. It interleaves GUI primitives with programmatic tool calls inside one action sequence. Training uses supervised finetuning first and then reinforcement learning so the model learns when a tool call is the safer choice. OSWorld success rises from 33.3% to 41.0%. The 7.7 point gain starts from a much weaker base than CoAct-1, which suggests that tool substitution matters most when the underlying model is least capable.

V-Droid (Dai et al., 2025) and VeriGUI (Zhang et al., 2026) keep every GUI step and harden it instead. V-Droid removes action generation from the pipeline. It extracts candidate actions from the UI tree and runs an LLM as a verifier that scores each one, and the highest scoring candidate executes. On AndroidWorld this raises success from 54.3% to 59.5% at 0.7 seconds per decision. The design rests on the assumption that discrimination is easier than generation, and the low latency confirms that the scoring pass is cheap. The limitation is the candidate set: if the extractor misses the correct action, no scoring function can recover it. VeriGUI verifies after each action instead of before it. It predicts the expected screen change for each step, compares the prediction to the actual screen, and retries with a corrected action on a mismatch. The reported figure is a 52.5% Recovery Success Rate, which is not task success. It measures the fraction of injected failure cases where the agent produces the correct recovery action, which means nearly half of injected failures go unrecovered.

The papers in this group agree that a naive click sequence is the wrong abstraction for long tasks. They disagree on where to spend effort. CoAct-1 and UltraCUA assume a large fraction of the work can bypass the GUI entirely, while V-Droid and VeriGUI assume it cannot and harden the GUI path instead. The two stances are complementary, but no paper here combines hybrid routing with per step verification on the remaining GUI actions, which is the obvious next experiment. The evidence base is also narrow. V-Droid reports on AndroidWorld, AndroidLab, and MobileAgentBench. UltraCUA and CoAct-1 report on OSWorld and WindowsAgentArena. VeriGUI reports on mobile benchmarks. UltraCUA shows out-of-domain generalization from Ubuntu to Windows, but the other methods do not test across platforms. VeriGUI assumes the post-action state is observable and stable, which breaks under animations, popups, and delayed loads, and it gives no breakdown of which error classes recover and which do not. The hybrid methods also carry a risk that the papers do not quantify. A misclick is usually reversible, but a wrong shell command or file write often is not. Neither CoAct-1 nor UltraCUA reports how often the code path causes irreversible state change, which is the failure that matters most once an agent has a terminal.

7.5 Skill Libraries and Training

The three methods in this group are CUA-Skill (Chen et al., 2026a), ClawGUI (Tang et al., 2026), and Gym-Anything (Aggarwal et al., 2026).

Most mitigation methods in this survey intervene at inference time by adding a planning pass, a verification loop, or a recovery branch onto a frozen model. Skill libraries and training frameworks take the opposite approach: they change what the agent already knows before execution begins. The failure modes they target are navigation drift and memory decay, and both are symptoms of the same underlying problem. An agent that reconstructs procedural knowledge from scratch on every run will take wrong turns and lose track of which substeps it has finished. An agent that already holds a verified route for a subtask avoids both problems, since the route encodes the menu path and the step ordering so neither has to be reinvented during execution.

CUA-Skill (Chen et al., 2026a) implements the library side of this idea. It stores parameterized Windows GUI skills as execution graphs, and at each step the agent retrieves candidate skills and composes them instead

of planning raw clicks. The authors report 57.5% under a best of three protocol on WindowsAgentArena, and in a single run ablation GPT-5 gains 15.6 points when given the skill library versus no library. That number needs interpretation, because best of three favors any stochastic policy and WindowsAgentArena rewards exactly the kind of fixed menu traversal that a skill graph encodes well. Even so, the core observation holds: most navigation errors are route errors rather than reasoning errors, and a cached route cannot take a wrong turn. The cost of this approach is maintenance, since skills become stale when the OS or the target application ships an update. The paper does not measure how fast the library degrades or what it takes to keep it current.

ClawGUI (Tang et al., 2026) and Gym-Anything (Aggarwal et al., 2026) address the same gap from the training side rather than the retrieval side. ClawGUI is an open source framework that runs online RL with dense process rewards on real devices. The dense rewards matter for memory failures specifically, because a sparse end of task signal cannot tell the agent which step drifted while a reward at every step can. The reported lift is 17.1% against an 11.1% baseline on MobileWorld, which is a real improvement but a low ceiling, and it is measured on mobile rather than desktop. Gym-Anything (Aggarwal et al., 2026) sits one layer back in the pipeline. It turns arbitrary software into a testable environment with automatically generated rubrics, which makes training data cheap enough for frameworks like ClawGUI to have something to learn from at scale. All three methods move beyond prompting a frozen general model, which suggests a shared view that prompting alone is insufficient for long-horizon GUI work. They differ on where the acquired knowledge should live: CUA-Skill (Chen et al., 2026a) externalizes it into a retrievable library, while ClawGUI (Tang et al., 2026) and Gym-Anything (Aggarwal et al., 2026) push it into the model weights. None of the surveyed papers tests whether the two approaches combine, even though a trained policy retrieving from a curated library appears to be a natural next step. None reports how its gains transfer across platforms or measures behavior after the environment changes underneath the agent, which is the scenario where long-horizon failures actually appear in deployment.

7.6 Cross-Cutting Observations

Methods cluster around a few established benchmarks. AndroidWorld (Rawles et al., 2024) and OSWorld (Xie et al., 2024) are the most common testbeds across the surveyed methods. Most methods report results on only two or three benchmarks. Four papers test their method only on their own benchmark. This concentration makes cross-method comparison easier on the established benchmarks but means the long-horizon benchmarks catalogued in Section 5 have little method evaluation, partly because they are too recent for methods to have caught up, as discussed in Section 10.

The mechanism categories are unevenly sized. Hierarchical has six methods, Context Management and Hybrid/Verifier have four each, Grounding and Skill/Training have three each. The uneven sizes reflect where the field has invested. Hierarchical decomposition is the most common design because it is cheap to add to an existing agent, while training-based approaches are rare because they need interaction data at scale.

8 Cross-Analysis: Failure Type Coverage

Section 6 showed which papers describe each failure type. This section asks a different question: of the twenty mitigation methods in Section 7, which ones are built mainly to fix each type? Each method paper was read and given a verdict of primary, secondary, or not addressed for each failure type. Primary means the abstract or introduction frames the method as a solution to that failure. Secondary means the paper mentions the failure as a benefit or side effect.

Figure 3 compares, for each failure type, the number of surveyed papers that describe it (blue bars) against the number of methods built mainly to fix it (red bars). The gap between the two bars is the coverage shortfall.

Figure 4 shows the underlying assignments. Each row is one of the twenty methods from Table 3. Each column is one of the ten failure types. A blue cell means that failure type is the method's primary target. A

Table 4: Failure type coverage by mitigation methods. Method names refer to entries in Table 3, where full citations are given.

Failure type	**Described in**	**Methods built mainly to fix it**	**Count**
Planning	8 papers	AgentProg, GTA1, Mobile-Agent-v3, NaturalGAIA, OS-Marathon, PLAN-AND-ACT, WebDART	7
Grounding	11 papers	CarePilot, CoAct-1, GTA1, Mobile-Agent-v3, RegionFocus, UGround	6
Accumulation	7 papers	CoAct-1, MobileUse, UltraCUA, VeriGUI	4
Memory	6 papers	AgentProg, AndroTMem, CarePilot, OCR-Memory	4
CrossApp	7 papers	AgentProg, AndroTMem, NaturalGAIA	3
Environment	7 papers	AgentProg, VeriGUI	2
Assumption	6 papers	MobileUse, VeriGUI	2
Navigation	10 papers	CUA-Skill	1
Forgetting	6 papers	AndroTMem	1
Instruction	7 papers	none	0

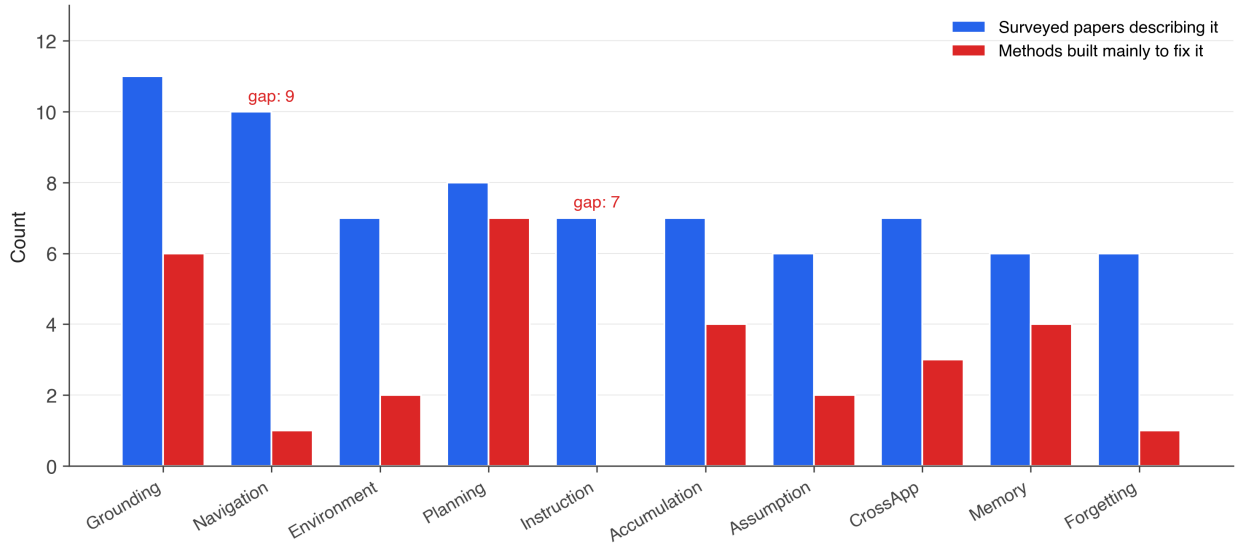

Figure 3: Failure type coverage: how often each type is described in the surveyed literature versus how many methods are built mainly to fix it.

yellow cell means the method addresses it as a secondary effect. The primary-count row at the top of each column matches the red bars in Figure 3.

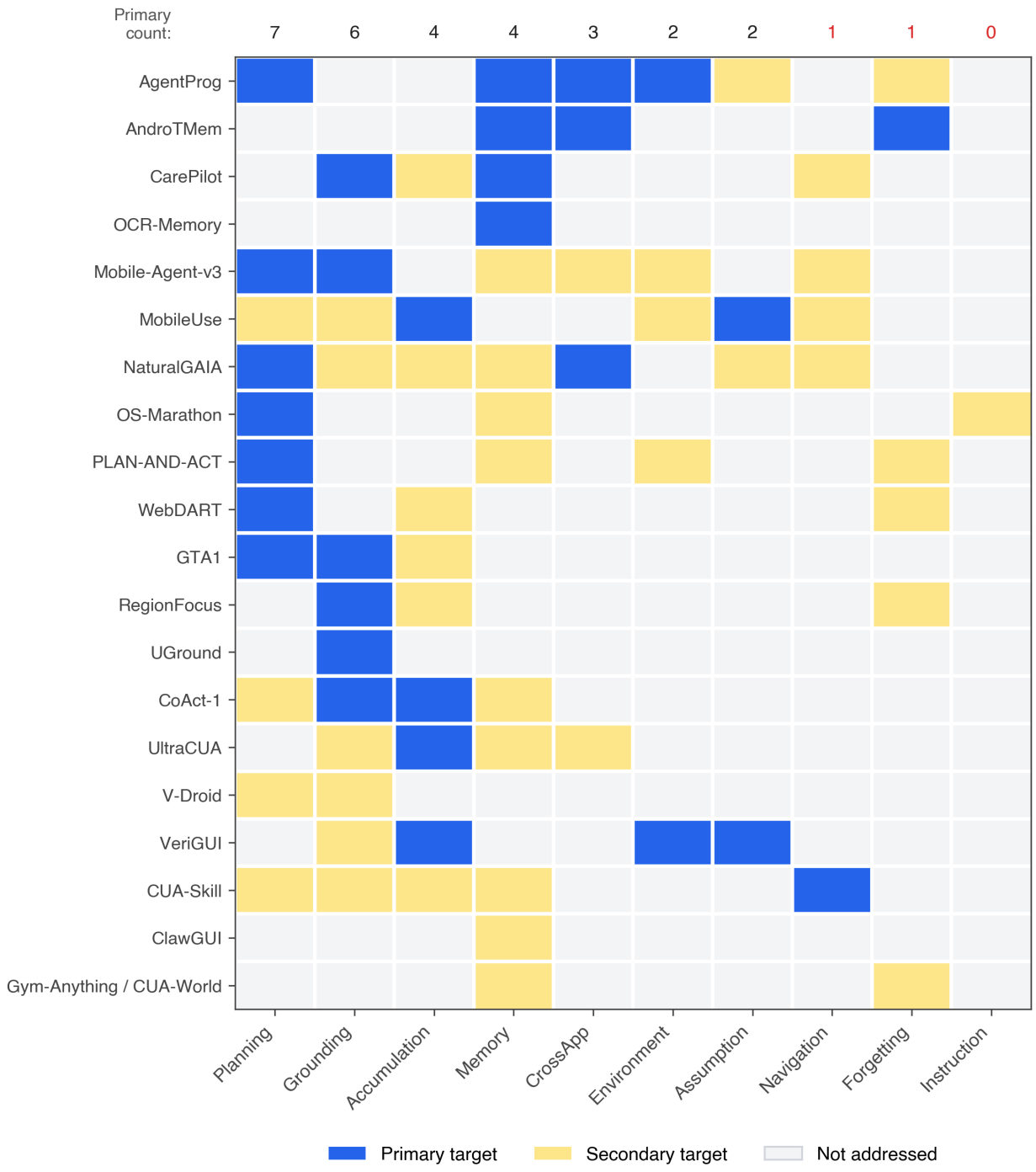

Figure 4: Method-level failure type coverage. Rows are methods (Table 3), columns are failure types (Table 2). Blue = primary target, yellow = secondary, gray = not addressed.

Three gaps stand out. Navigation is described in ten surveyed papers. We did not find a method explicitly designed to target Navigation as its main contribution other than CUA-Skill (Chen et al., 2026a). Instruction

is described in seven surveyed papers. We did not find any method explicitly designed to target Instruction as its main contribution. Forgetting is described in six surveyed papers but has one dedicated method, AndroTMem (Shi et al., 2026a).

By contrast, Planning and Grounding each have six or seven dedicated methods. The field has concentrated effort on failures that occur within a single step or sub-plan and has put little effort into failures that span steps, such as forgetting constraints stated earlier, navigating unfamiliar application conventions, or correctly interpreting the task instruction.

This gap matters because of the success-rate observations in Section 5.4. On the hardest surveyed benchmarks, current agents fail most of the time, so the under-addressed failure types do not need to be common to be worth fixing. A method that addresses a failure type responsible for even a modest share of those failures would produce one of the largest available single improvements. The benchmark catalog in Section 5 also shows the field moving toward exactly the tasks where cross-step failures are likely: long workflows, multi-application tasks, real user instructions with ambiguous references. Benchmarks are increasingly built to stress failure types that few methods are designed to fix. The imbalance indicates that the field has prioritized improving local decision accuracy over addressing cross-step reliability, which the benchmark literature suggests is where long-horizon failures concentrate.

9 Case Study: Secondary Analysis of Published Results

The cross-analysis in Section 8 is based on what papers report. This section validates the catalog by treating it as a dataset and running two analyses on the published numbers. No new agent runs were performed. The case study does not introduce new metrics or new contributions. It is a secondary analysis, not a meta-analysis, because the underlying studies use different base models, different task distributions, and different evaluation protocols, which rules out the effect-size pooling that a formal meta-analysis requires. The results below should therefore be interpreted as patterns in the reported literature rather than as controlled comparisons.

9.1 Benchmark Ranking Agreement

If the twenty-five benchmarks all measure the same underlying capability, they should rank the same agents in the same order. We extracted the published results table from each benchmark paper, normalized agent names, and kept agents reported on at least two benchmarks. Fourteen benchmarks report leaderboards with usable agent identities, and twenty-one agent configurations appear on two or more benchmarks. We kept an entry only when the paper evaluates the base model or a clearly identified configuration, and we treated framework-wrapped or finetuned variants as distinct entries. For each pair of benchmarks with at least three common agents, we computed Kendall's tau rank correlation.

Fifteen benchmark pairs have at least three common agents. Figure 5 shows the pairwise correlation.

The mean tau across pairs is 0.22 and the median is 0.33. Four of fifteen pairs have negative tau, meaning those benchmark pairs rank the same agents in opposite order. Five of fifteen pairs have tau of 0.80 or higher. Sample sizes per pair are small (three to five common agents), so individual tau values are noisy and none reach statistical significance. With three common agents a tau of 1.00 or -1.00 can arise from a single chance ordering, so we read the distribution across pairs rather than any single value. The pattern across pairs is the finding: agreement is weak and inconsistent, with some benchmark pairs in strong agreement and others in outright disagreement.

This weak agreement is consistent with the Section 5 observation that benchmarks define horizon differently, use different evaluation methods, and test different task distributions. An agent that wins on one long-horizon benchmark is not reliably better on another. Published claims that a method is state of the art on one benchmark carry less signal than they appear to.

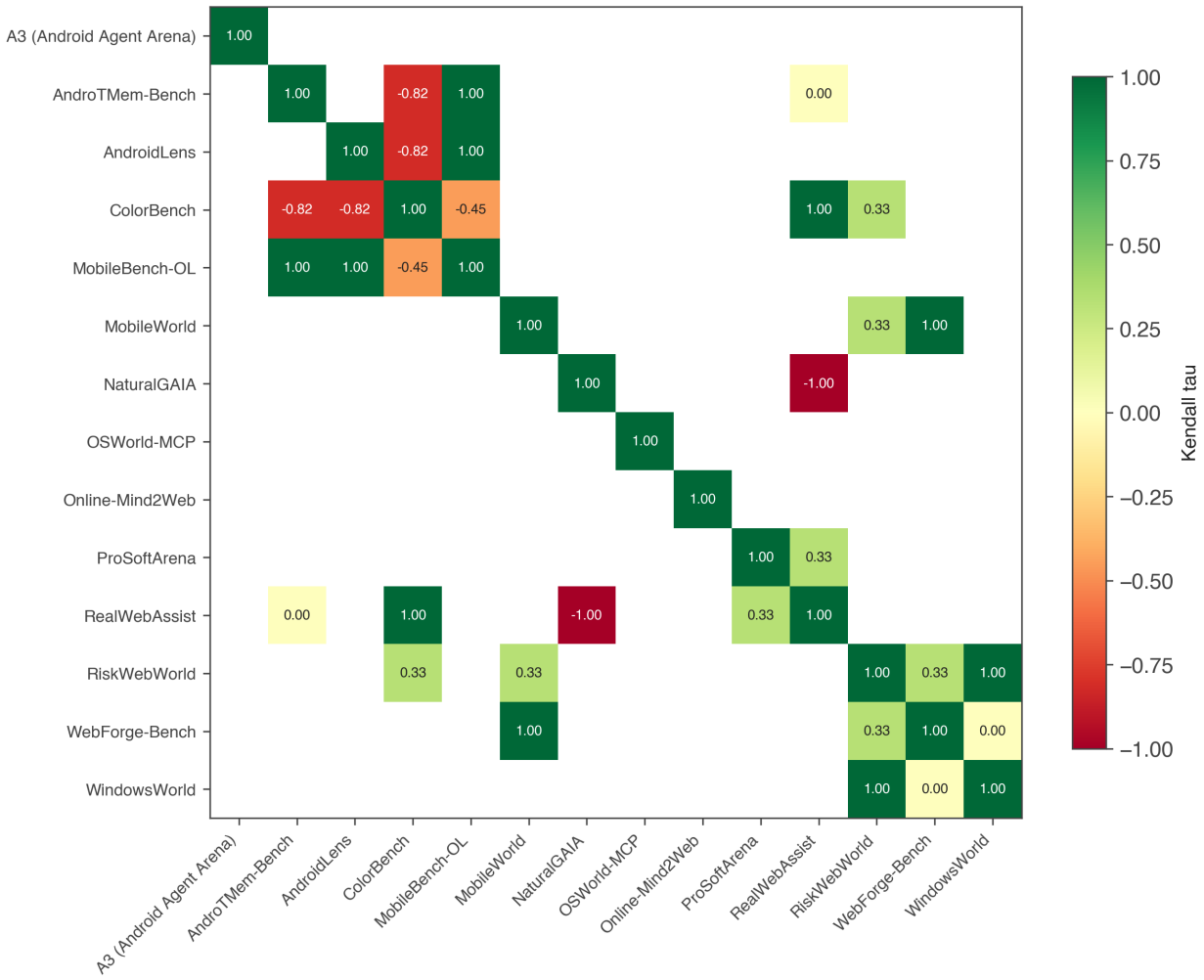

Figure 5: Benchmark ranking agreement (Kendall tau). Blank cells have fewer than three common agents.

9.2 Method Gain vs. Benchmark Horizon

For each of the twenty methods in Section 7 we extracted the reported gain over its baseline, the benchmark where that gain was reported, and the average horizon of that benchmark. Figure 6 shows the result.

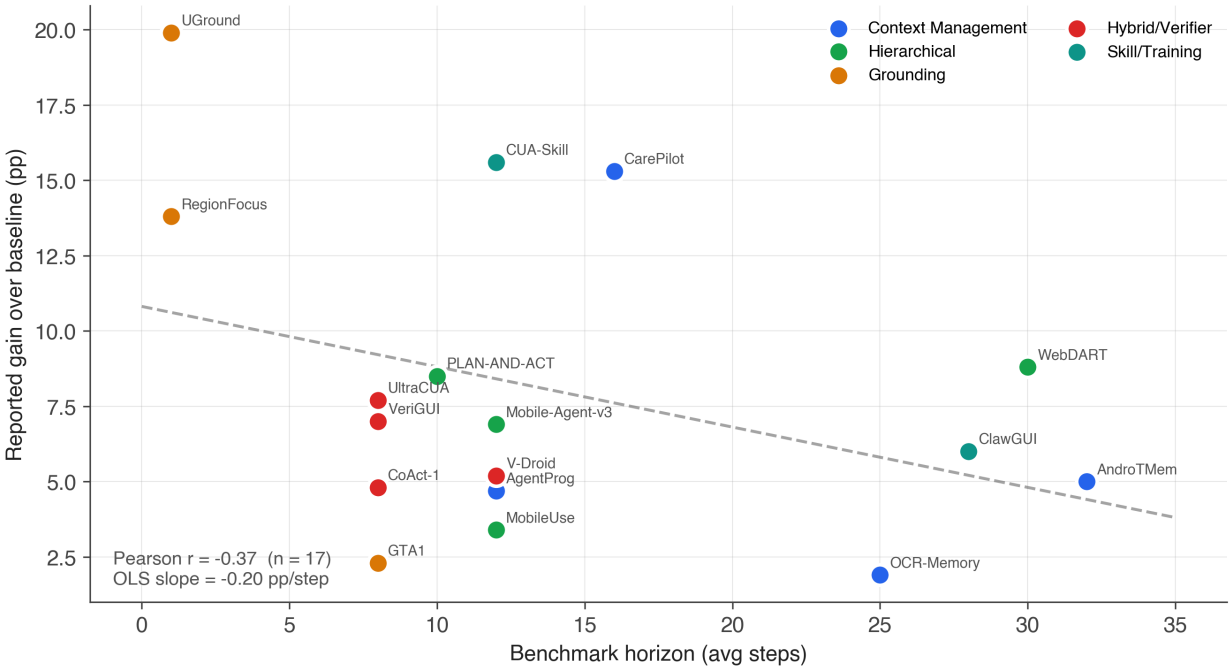

Figure 6: Method gain vs. benchmark horizon by mechanism category.

Reported gains decrease as benchmark horizon increases. The Pearson correlation between reported gain and benchmark horizon is -0.37 across seventeen method-benchmark pairs, with an OLS slope of -0.20 percentage points per step. Methods that report large gains mostly tested on short-horizon benchmarks. The two methods with the largest gains, UGround (Gou et al., 2024) at 20 points and RegionFocus (Luo et al., 2025) at 14 points, both tested primarily on single-step grounding benchmarks. Methods tested on benchmarks averaging twenty-five steps or more report gains of two to nine points.

By mechanism category, Grounding methods show the highest mean gain at twelve points, but this is confounded: Grounding methods are tested on grounding benchmarks where their mechanism addresses the whole task. Hierarchical and Context Management methods show similar mean gains of about seven points each on longer benchmarks. Skill/Training methods show the highest gain on a long benchmark, CUA-Skill (Chen et al., 2026a) at fifteen points on WindowsAgentArena (Bonatti et al., 2024).

Three interpretations are possible and they are not mutually exclusive. First, the easy wins on short horizons have been taken, and the remaining gains on long horizons are genuinely smaller. Second, methods default to the established short-horizon benchmarks because those benchmarks have stable harnesses, known baselines, and prior results to compare against. Third, the long-horizon benchmarks are too new for methods to have been tested on them, which the Section 5 observation on method coverage supports. The data here cannot separate these three causes. What it does show is that as of the literature cutoff, the published evidence about method effectiveness is concentrated at the short end of the horizon axis, which is the opposite of where the open problems are.

9.3 Future Work: Failure Type Distribution

A second validation measures how often each of the Section 6 failure types occurs in practice. Two agents run on two benchmarks. An LLM judge classifies each failed trajectory into a Section 6 type using the

definitions as the prompt, validated against a human-labeled subset. This was not executed for the present paper and is listed as future work.

10 Limitations

This survey has six limitations that readers should weigh when interpreting the findings.

The failure type coding and mechanism grouping are single-coder. The failure type verdicts in Table 4 and the mechanism categories in Table 3 were both assigned by one coder using the rules stated in Section 4.4. The full coding table with supporting quotes is provided as supplementary material. Inter-rater reliability was not measured. A second independent coder could disagree with some verdicts or category assignments, which would change the counts in Table 4 and the groupings in Section 7. We consider the Navigation and Instruction gaps reliable because they are large: a verdict change in one or two papers would not close a gap of nine or eight. The smaller differences between Planning, Grounding, and Accumulation are less reliable and should not be over-interpreted. The mechanism categories are descriptive groupings for discussion, not a theoretical proposal.

Our secondary analysis treats published numbers as comparable when they may not be. Section 9 aggregates success rates and gains reported by different papers on different benchmarks with different base models, prompts, step budgets, and evaluation dates. An agent that scores 40 percent on one benchmark and 20 percent on another may differ for reasons unrelated to the benchmarks. In addition, several benchmarks run on live websites or proprietary applications whose behavior changes over time, which Online-Mind2Web (Xue et al., 2025) documents directly, so two papers reporting scores on the same benchmark six months apart may have measured different environments. Our ranking-agreement analysis in Section 9.1 does not control for any of these confounds, and the correlations in Section 9 should be read as patterns in the published literature, not as controlled comparisons.

Our horizon axis uses inconsistent units. The gain-versus-horizon regression in Section 9.2 places every method on an x-axis of benchmark horizon measured in steps, but as Section 5.4 observes, different benchmarks count different things. Some count agent actions, some count human reference steps, some count sub-goal checkpoints. We mapped each to an approximate step count, which introduces measurement error into the regression. The negative correlation we report is a real pattern in the published numbers, but its magnitude is less precise than the figure implies.

Most sources are preprints. Of the 41 in-scope papers, the majority are arXiv preprints that had not completed peer review at the literature cutoff. This reflects publishing norms in a fast-moving area of machine learning, where results appear on arXiv months or years before journal or conference publication. Numbers, claims, and benchmark definitions may change between the preprint and a final version. Readers who need a precise result should check the current arXiv version of the cited paper.

The literature cutoff introduces survivorship bias. The cutoff is April 2026. The field is moving fast. Benchmarks and methods published after the cutoff are not covered, and some findings may already be stale by the time this survey appears. The success-rate observations in Section 5.4 are at-publication numbers, and the established short-horizon benchmarks moved from roughly 12 percent to over 60 percent in eighteen months (Xie et al., 2024; Song et al., 2025a), so the long-horizon scores reported here may rise on a similar trajectory. The structural findings about failure-type coverage and benchmark coverage are less time-sensitive, since adding a method that targets a new failure type requires new research rather than a stronger base model. As a concrete indicator of pace, our scan of arXiv between the cutoff and submission found at least four new GUI agent benchmarks and two adjacent surveys (Hu et al., 2026; Qi et al., 2026) in two months. We recommend checking for competing or superseding work before acting on the coverage gaps reported here.

Method evaluation on new benchmarks lags publication. The longest-horizon benchmarks, including WindowsWorld (Li et al., 2026b) and CUA-World (Aggarwal et al., 2026), were published in April 2026, at the same time as or after most of the surveyed methods. No method could have been tested on them by the cutoff. This is a natural consequence of a fast-moving field and limits what the survey can say about

method effectiveness at the longest horizons. As the field matures and methods accumulate results on the newer benchmarks, the coverage picture will change.

11 Open Problems

The catalog and cross-analysis point to six open problems.

Standardize the long-horizon definition. Most benchmarks measure horizon in steps, but the counted unit varies: some count agent actions, some count human reference steps, some count milestones or sub-goal checkpoints, and some evaluate at multiple fixed step budgets. Two benchmarks, ST-WebAgentBench (Levy et al., 2024) and WebChoreArena (Miyai et al., 2025), do not report a step count and characterize horizon qualitatively. Until the field agrees on what a step is, cross-benchmark comparison will remain unreliable.

Test methods on the new benchmarks. The longest-horizon benchmarks (CUA-World (Aggarwal et al., 2026), OS-Marathon (Wu et al., 2026a), WindowsWorld (Li et al., 2026b)) have at most one external method tested on them. Methods that work on AndroidWorld may not transfer to tasks ten times longer.

Address the Navigation gap. Ten surveyed papers describe agents failing because they do not know where a function lives in an application. CUA-Skill (Chen et al., 2026a) builds a skill library. Other approaches, such as application documentation retrieval or structured UI exploration, are unexplored.

Address the Instruction gap. Eight surveyed papers describe agents misinterpreting the task instruction. No surveyed method is built mainly to fix this. RealWebAssist (Ye et al., 2025b) shows that real user instructions are ambiguous and context-dependent, which makes this gap worse as deployments move to real users.

Report failure distributions, not just success rates. When a benchmark reports that the best agent succeeds on 8 percent of tasks, a method developer learns that the problem is hard but not where to start. When a benchmark reports that failures break down as 40 percent grounding errors, 25 percent navigation errors, and 20 percent instruction misreads, the developer knows exactly where the largest improvement is. Two surveyed benchmarks already move in this direction: NaturalGAIA (Zheng et al., 2025) attributes each failure to a specific atomic step in its causal pathway, and OS-HARM (Kuntz et al., 2025) labels unsafe actions by type. HORIZON (Wang et al., 2026) does the same for cross-domain agentic systems with a diagnostic taxonomy. The other benchmarks in the catalog report a low number without telling anyone why it is low. Reporting a failure distribution alongside the success rate should become standard practice. The vocabulary in Section 6 is one candidate codebook.

Understand the trade-off between Memory and Forgetting as context grows. Section 6 distinguishes Memory failures, where the constraint was truncated out of context, from Forgetting failures, where the constraint is still in context but the agent ignores it. As context windows grow, fewer failures should be attributable to Memory and more to Forgetting, since the constraint stays in the prompt but the agent must attend to it among more distractors. The surveyed context-management methods in Section 7.1 restructure history rather than simply rely on larger windows, which is an implicit bet that restructuring helps where expansion does not. We did not find a paper that measures the trade-off directly. Whether larger windows help or hurt is important for deciding where to invest: in context management, in attention-improving training, or in context expansion.

Measure recovery and reversibility. We did not find a surveyed benchmark that measures whether an agent can detect and recover from its own mistakes, or whether its mistakes leave the environment in a recoverable state. For agents deployed with real user data, a wrong click that silently deletes a file is worse than a wrong click that fails loudly. This is left to future work.

12 Conclusion

We surveyed twenty-five long-horizon GUI agent benchmarks and twenty mitigation methods published between 2024 and 2026. We normalized failure terminology into a vocabulary of ten types and mapped which methods are built to fix each. The benchmark landscape is converging on multi-application, long-

step, real-instruction tasks. The method landscape is concentrated on within-step failures. Three cross-step failure types that benchmarks stress heavily (Navigation, Forgetting, Instruction) have one or zero dedicated methods.

The secondary analysis shows that this mismatch is reflected in published results. Benchmarks agree only weakly on which agents are best, and reported method gains shrink as benchmark horizon grows. The published evidence about what works is concentrated on short, established benchmarks, while the benchmarks that measure the hard problems are too new to have been tested.

GUI agents will be deployed on systems that hold real user data and control real processes. Their reliability at long horizons is not a leaderboard question. It is a systems engineering question, and the field does not yet have the evaluation infrastructure to answer it. Closing the Navigation, Forgetting, and Instruction gaps, standardizing what long horizon means, and testing methods on the benchmarks where they are hardest to win rather than easiest is the clearest path forward.

References

Abhyankar et al. Osworld-human: Benchmarking the efficiency of computer-use agents. *arXiv preprint arXiv:2506.16042*, 2025.

Aggarwal et al. Gym-anything: Turn any software into an agent environment. *arXiv preprint arXiv:2604.06126*, 2026.

Ai et al. Prosoftarena: Benchmarking hierarchical capabilities of multimodal agents in professional software environments. *arXiv preprint arXiv:2601.02399*, 2026.

J. Bai et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Bedi et al. Healthadminbench: Evaluating computer-use agents on healthcare administration tasks. *arXiv preprint arXiv:2604.09937*, 2026.

Bonatti et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

Cao et al. Androidlens: Long-latency evaluation with nested sub-targets for android gui agents. *arXiv preprint arXiv:2512.21302*, 2025.

Chai et al. A3: Android agent arena for mobile gui agents with essential-state procedural evaluation. *arXiv preprint arXiv:2501.01149*, 2025.

Chen et al. Cua-skill: Develop skills for computer using agent. *arXiv preprint arXiv:2601.21123*, 2026a.

Chen et al. Riskwebworld: A realistic interactive benchmark for gui agents in e-commerce risk management. *arXiv preprint arXiv:2604.13531*, 2026b.

Y. Chen et al. A survey on benchmarks of llm-based gui agents. *TechRxiv*, 2026c.

K. Cheng et al. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proc. ACL*, 2024.

Dai et al. Advancing mobile gui agents: A verifier-driven approach to practical deployment. *arXiv preprint arXiv:2503.15937*, 2025.

Google DeepMind. Gemini 2.5 technical report. 2025.

X. Deng et al. Mind2web: Towards a generalist agent for the web. In *Proc. NeurIPS*, 2023.

Erdogan et al. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*, 2025.

Ghosh et al. Carepilot: A multi-agent framework for long-horizon computer task automation in healthcare. *arXiv preprint arXiv:2603.24157*, 2026.

G. Gonzalez-Pumariega et al. On the reliability of computer use agents. *arXiv preprint arXiv:2604.17849*, 2026.

Gou et al. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

Gou et al. Mind2web 2: Evaluating agentic search with agent-as-a-judge. *arXiv preprint arXiv:2506.21506*, 2025.

H. He et al. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proc. ACL*, 2024.

W. Hong et al. Cogagent: A visual language model for gui agents. In *Proc. CVPR*, 2024.

Hu et al. Memory in the age of ai agents: A survey. *arXiv preprint arXiv:2512.13564*, 2025.

J. Hu et al. Gui agents with reinforcement learning: Toward digital inhabitants. *arXiv preprint arXiv:2604.27955*, 2026.

Jang et al. Odysseys: Benchmarking web agents on realistic long horizon tasks. *arXiv preprint arXiv:2604.24964*, 2026.

Jia et al. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents. *arXiv preprint arXiv:2510.24563*, 2025.

R. Kapoor et al. Omniact: A dataset and benchmark for enabling multimodal generalist agents for desktop and web. In *Proc. ECCV*, 2024.

J. Y. Koh et al. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proc. ACL*, 2024.

Kong et al. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025.

Kuntz et al. Os-harm: A benchmark for measuring safety of computer use agents. *arXiv preprint arXiv:2506.14866*, 2025.

Levy et al. St-webagentbench: A benchmark for evaluating safety and trustworthiness in web agents. *arXiv preprint arXiv:2410.06703*, 2024.

Li et al. Mobileuse: A gui agent with hierarchical reflection for autonomous mobile operation. *arXiv preprint arXiv:2507.16853*, 2025a.

Li et al. Ocr-memory: Optical context retrieval for long-horizon agent memory. *arXiv preprint arXiv:2604.26622*, 2026a.

Li et al. Windowsworld: A process-centric benchmark of autonomous gui agents in professional cross-application environments. *arXiv preprint arXiv:2604.27776*, 2026b.

K. Li et al. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*, 2025b.

Luo et al. Visual test-time scaling for gui agent grounding. *arXiv preprint arXiv:2505.00684*, 2025.

Miyai et al. Webchorearena: Evaluating web browsing agents on realistic tedious web tasks. *arXiv preprint arXiv:2506.01952*, 2025.

M. Mohammadi et al. Evaluation and benchmarking of llm agents: A survey. In *Proc. ACM SIGKDD*, 2025.

OpenAI. Gpt-4v(ision) system card. 2023.

J. Qi et al. Towards trustworthy agentic ai: A comprehensive survey of safety, robustness, privacy, and system security. *arXiv preprint arXiv:2605.23989*, 2026.

Y. Qin et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Rawles et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

C. Rawles et al. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023.

P. Sager et al. A comprehensive survey of agents for computer use. *arXiv preprint arXiv:2501.16150*, 2026.

Shi et al. Androtmem: From interaction trajectories to anchored memory in long-horizon gui agents. *arXiv preprint arXiv:2603.18429*, 2026a.

Y. Shi et al. Towards trustworthy gui agents: A survey. *arXiv preprint arXiv:2503.23434*, 2026b.

Song et al. Coact-1: Computer-using multi-agent system with coding actions. *arXiv preprint arXiv:2508.03923*, 2025a.

Song et al. Colorbench: Benchmarking mobile agents with graph-structured framework for complex long-horizon tasks. *arXiv preprint arXiv:2510.14621*, 2025b.

Tang et al. Clawgui: A unified framework for training, evaluating, and deploying gui agents. *arXiv preprint arXiv:2604.11784*, 2026.

Tian et al. Agentprog: Empowering long-horizon gui agents with program-guided context management. *arXiv preprint arXiv:2512.10371*, 2025.

Wang et al. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*, 2026.

J. Wang et al. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024.

W. Wang et al. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025.

Wu et al. Os-marathon: Benchmarking computer-use agents on long-horizon repetitive tasks. *arXiv preprint arXiv:2601.20650*, 2026a.

Wu et al. Mobilebench-ol: A comprehensive chinese benchmark for evaluating mobile gui agents in real-world environment. *arXiv preprint arXiv:2601.20335*, 2026b.

Z. Wu et al. Os-atlas: A foundation action model for generalist gui agents. In *Proc. ICLR*, 2025.

T. Xie et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Proc. NeurIPS*, 2024.

Y. Xu et al. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

Xue et al. An illusion of progress? assessing the current state of web agents. In *Proc. COLM*, 2025.

Yang et al. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*, 2025a.

Yang et al. Ultracua: A foundation model for computer use agents with hybrid action. *arXiv preprint arXiv:2510.17790*, 2025b.

Yang et al. Webdart: Dynamic decomposition and re-planning for complex web tasks. *arXiv preprint arXiv:2510.06587*, 2025c.

Ye et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025a.

Ye et al. Realwebassist: A benchmark for long-horizon web assistance with real-world users. *arXiv preprint arXiv:2504.10445*, 2025b.

K. You et al. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *Proc. ECCV*, 2024.

Yuan et al. Webforge: Breaking the realism-reproducibility-scalability trilemma in browser agent benchmark. *arXiv preprint arXiv:2604.10988*, 2026.

Zhang et al. Don't act blindly: Robust gui automation via action-effect verification and self-correction. *arXiv preprint arXiv:2604.05477*, 2026.

C. Zhang et al. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*, 2024.

C. Zhang et al. Llm-brained gui agents: A survey. *Transactions on Machine Learning Research*, 2025.

Zheng et al. Naturalgaia: A verifiable benchmark and hierarchical framework for long-horizon gui tasks. *arXiv preprint arXiv:2508.01330*, 2025.

S. Zhou et al. Webarena: A realistic web environment for building autonomous agents. In *Proc. ICLR*, 2024.