The Reliability Gap: Why Capable Computer-Use Agents Still Fail Twice

Anonymous ACL submission

Abstract

Computer-use agents now match or exceed humans on benchmarks like OSWorld, yet the same agent on the same task can succeed once and fail the next. The gap is large: a strong agent reaches **78% Pass@10** but only **36% Pass^10** (the rate at which all 10 attempts succeed), so capability is more than twice what reliability delivers. We decompose this *reliability gap* into stochasticity, instruction ambiguity, and planning variability, probing each with controlled interventions across six frontier and open-source computer-use agents on OSWorld and evaluating with paired statistical tests over repeated executions. Three findings stand out: (1) deterministic decoding does not consistently improve reliability and can regress one model by 20 tasks; (2) clarifying instructions to expose evaluator-aligned criteria yields larger and more consistent gains than removing sampling noise; and (3) a user-simulator providing targeted feedback on failed runs recovers up to **+28 pp Pass^3**, near-doubling reliability for one model and exceeding every static intervention. We also propose our evaluation recipe, Pass^k combined with McNemar and Wilcoxon signed-rank tests over per-task outcomes, as a reusable protocol for CUA reliability studies. Reliable computer-use agents will not be built by chasing determinism, but by treating ambiguity as irreducible and resolving it interactively, and by favoring strategies that remain stable across runs.

1 Introduction

Consider an OSWorld task: rename every PNG in a folder to lowercase. Run 1 succeeds; run 2 fails because the agent reaches for a file manager that skips hidden files; run 3 fails on a context-menu click that lands one pixel off; run 4 succeeds again. The task, model, and prompt are unchanged; only behavior across runs varies. This is the everyday reality of computer-use agents: capable in expectation, but not consistently so.

This gap is invisible in the metrics currently used to report progress. On OSWorld (Xie et al., 2024), modern agents outperform human baselines (Anthropic, 2026; Gonzalez-Pumariega et al., 2026; Bai et al., 2026), but results are typically reported as single-run, averaged, or Best-of-N performance, all of which reward succeeding *once*. As Figure 1 (left) shows, a strong agent (Agent S3 with GPT-5) reaches **Pass@10** $\approx$ **78%** yet **Pass^10** $\approx$ **36%** (the fraction of tasks for which all 10 attempts succeed): more than half of apparent capability disappears once we require reproducibility. We call this the *reliability gap*. Safety-critical software (Parasoft Corporation, 2023; Eaton Corporation, 2019; ROHM Semiconductor, 2020; Intuition Labs, 2022) and regulated AI systems (Tabassi, 2023; AI HLEG, 2019) demand reproducible behavior to several nines, and current agents are an order of magnitude away from those bars. **If an agent can succeed at a task once, what prevents it from doing so reliably?**

To answer this question, we decompose task execution into three components (Figure 1, right): task specification, agent decision-making, and execution stochasticity. Each admits a controlled intervention: deterministic decoding and environment perturbation probe stochasticity; instruction clarification and a user-simulator probe ambiguity; plan extraction and refinement probe planning variability. We run these on OSWorld with frontier (GPT-5, Claude Sonnet 4.6, Kimi 2.5) and open-source (Qwen-3VL-8B, OpenCUA, UI-TARS-1.5) agents. To measure reliability properly we pair Pass^k (Yao et al., 2024) with paired statistical tests over per-task outcomes (McNemar for transitions in and out of the reliably-solved set, Wilcoxon signed-rank for shifts in per-task success counts), detecting both improvements and regressions at the task level.

We observe three surprising findings:

- **Determinism is not the answer.**

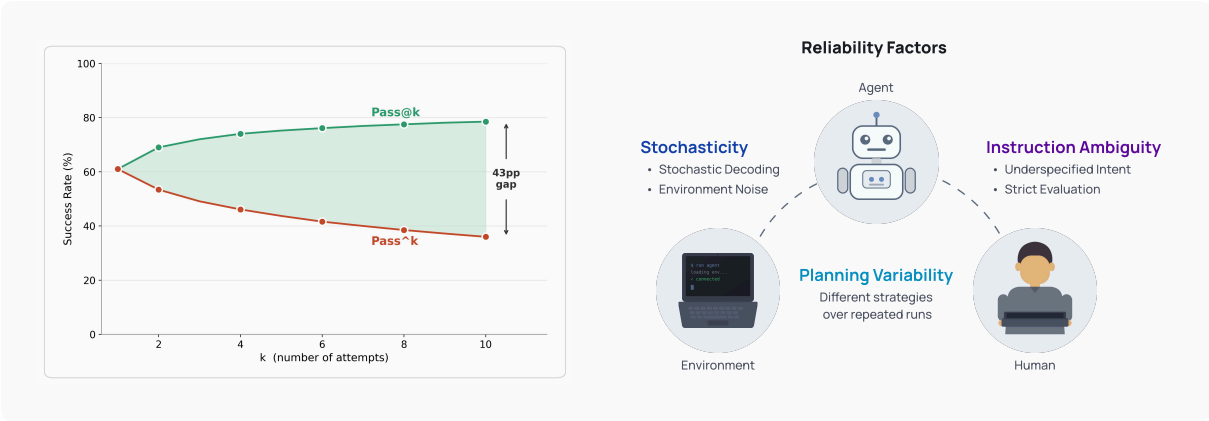


Figure 1: (left) Performance of a strong computer-use agent (Agent S3 with GPT-5) across repeated attempts. While Pass@10 reaches approximately 78%, the corresponding Pass^10 indicates that the agent succeeds on all 10 executions for only about 36% of tasks, indicating that achieving reliability across repeated executions is challenging. (right) Overview of factors that contribute to unreliability. We decompose task execution into three components: stochasticity during execution, ambiguity in task specification, and variability in agent behavior, which adversely affect reliability across repeated executions.

Temperature-0 batch-invariant decoding with a fixed high-level plan does not consistently improve reliability. Qwen regresses by 20 tasks ($b - c = -20$); OpenCUA and UI-TARS improve. Sampling noise is not the dominant axis of unreliability.

- **Specification dominates randomness.** Clarifying instructions to expose evaluator-aligned criteria yields significant gains across every frontier model (GPT-5: $b - c = 48$; Claude: $b - c = 22$; Kimi: $\Delta c_x = 0.141$), substantially larger than any determinism intervention. A significant share of unreliability is a specification problem the evaluation methodology has been silently tolerating.

- **Interaction is the strongest single lever.** A user-simulator providing targeted feedback on failed runs lifts Kimi by $+28$ pp on Pass^3 (a near-doubling of reliability) and outperforms blind retries on the same budget across every frontier model, isolating the gain to the feedback content rather than additional attempts.

These results imply that CUA reliability will come less from making agents *more deterministic* and more from making tasks *less ambiguous* (interactively, since ambiguity is irreducible) and strategies *more stable*. Reporting Pass@k without Pass^k will continue to overstate capability by more than $2\times$ for one of our strongest agent.

2 CUA Reliability Evaluation

To study how different sources of variation affect reliability across repeated executions, we aim to answer the following research questions:

1. **Stochasticity**: To what extent does stochasticity in decoding and environment dynamics contribute to unreliable outcomes across repeated executions?

2. **Instruction Ambiguity**: To what extent does ambiguity in task specification, and its mismatch with evaluation criteria, contribute to reliability?

3. **Planning Variability**: To what extent does variability in agent strategies across executions drive unreliability, beyond effects due to instruction ambiguity?

2.1 Problem Formulation

To analyze reliability across repeated executions of the same task, we formalize computer-use agents (CUAs) within a POMDP framework. Specifically, we model task execution as a partially observable Markov Decision Process (POMDP) defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{I}, R \rangle$, where $\mathcal{S}$ is the state space encoding the computer state, $\mathcal{O}$ is the observation space such as desktop screenshots, $\mathcal{A}$ is the action space of the agent (e.g. `agent.click(...)` and `agent.type(...)`), $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a stochastic transition function, $\mathcal{I}$ is the space of possible user instructions represented in natural

language, and $R : (\mathcal{S} \times \mathcal{A})^* \times \mathcal{I} \to [0, 1]$ denotes the instruction reward function that assigns a scalar reward to a trajectory $\tau := (s_0, a_0, \ldots, a_{T-1}, s_T)$ conditioned on instruction $I \in \mathcal{I}$. A task is defined as $x := (s_0, I) \in \mathcal{X}$, consisting of an initial state and instruction.

Standard approaches optimize for expected success, maximizing $\mathbb{E}_{\tau\sim\pi(\cdot|x)}[R(\tau, x)]$, where $R(\tau, x) \in [0, 1]$ denotes a scalar reward for a single execution of task x. For simplicity in evaluating reliability, we consider a binary success outcome $r_{x,j} = R(\tau_j, x)$ such that $r_{x,1}, \ldots, r_{x,n} \in \{0, 1\}$ denote the outcomes of n executions of the policy on the same task x.

Under this formulation, success corresponds to the probability that a single execution succeeds, $\mathbb{E}_{\tau\sim\pi(\cdot|x)}[r_{x,1}] = \Pr(r_{x,1} = 1)$. In contrast, we define reliability as the probability that all executions of the same task succeed, given by $\mathbb{E}_{\tau_1,\ldots,\tau_n\sim\pi(\cdot|x)}\left[\prod_{j=1}^{n} r_{x,j}\right] = \Pr(r_{x,1} = 1, r_{x,2} = 1, \cdots, r_{x,n} = 1)$.

2.2 Reliability Metrics

Given the definition of reliability as reproducible success across repeated executions, we seek metrics that characterize how consistently an agent succeeds on the same task. For each task x, we execute the policy n times, yielding binary outcomes $r_{x,1}, \ldots, r_{x,n} \in \{0, 1\}$ and $c_x = \sum_{j=1}^{n} r_{x,j}$ successful runs. These repeated outcomes allow us to categorize tasks as *consistently solved* ($c_x = n$), *inconsistently solved* ($0 < c_x < n$), or *never solved* ($c_x = 0$), providing a structured view of variability across runs.

Pass^k (Repeated-Run Success). Following (Yao et al., 2024), we adopt the metric

$$\text{Pass^k} = \mathbb{E}_{x\sim\mathcal{X}}\left[\frac{\binom{c_x}{k}}{\binom{n}{k}}\right],$$

which estimates the probability that k executions of a task all succeed. We highlight two cases: (1) Pass^1, the marginal success rate across executions, given by Pass^1 $= \mathbb{E}_{x\sim\mathcal{X}}\left[\frac{c_x}{n}\right]$, which captures agent capability and (2) Pass^n, the fraction of tasks that succeed on all repeated executions, given by Pass^n $= \mathbb{E}_{x\sim\mathcal{X}}\left[\mathbf{1}[c_x = n]\right]$, which directly measures reliability as reproducible success.

While Pass^k summarizes performance across tasks, it does not capture how outcomes change on the same tasks across different settings. To study how reliability varies across settings and identify improvements or regressions at the task level, we use paired statistical tests.

McNemar Test (Reliability Transitions). We define a binary indicator $z_x = \mathbf{1}[c_x = n]$ for whether task x is consistently solved. To compare two settings, we apply McNemar's test (McNemar, 1947) to paired outcomes $\{z_x^{(\text{base})}, z_x^{(\text{new})}\}$. Let

$$b = \sum_x \mathbf{1}[z_x^{(\text{base})} = 0,\ z_x^{(\text{new})} = 1],$$

$$c = \sum_x \mathbf{1}[z_x^{(\text{base})} = 1,\ z_x^{(\text{new})} = 0]$$

denote the number of tasks that improve and regress, respectively. The test statistic is

$$\chi^2 = \frac{(b - c)^2}{b + c},$$

which measures the imbalance between improvements and regressions. This test evaluates whether more tasks are reliably solved than not. We report $(b - c)$, which indicates the direction of change, and compute p-values using the corresponding χ^2 statistic, with significance assessed at $p < 0.05$.

Wilcoxon Signed-Rank Test (Consistency Improvements). To capture partial improvements in repeated-run performance, we compare per-task success counts across settings. For each task, we compute differences $d_x = c_x^{(\text{new})} - c_x^{(\text{base})}$ and apply the Wilcoxon signed-rank test (Wilcoxon, 1945). Let $\{d_x\}$ denote the set of nonzero differences, and let R_x be the rank of $|d_x|$ among these values (with ties assigned average ranks). The test statistic is

$$W = \sum_{d_x > 0} R_x,$$

which measures whether improvements tend to be larger or more frequent than regressions. This detects incremental improvements in consistency (e.g., $1 \to 2$ or $2 \to 3$ successes), even when full reproducibility is not achieved. We report $\Delta c_x = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} d_x$, the average change in per-task success counts, and compute p-values from the Wilcoxon signed-rank test using the test statistic W, with significance assessed at $p < 0.05$.

2.3 Models

We evaluate reliability across a range of computer-use agents spanning both frontier and open-source

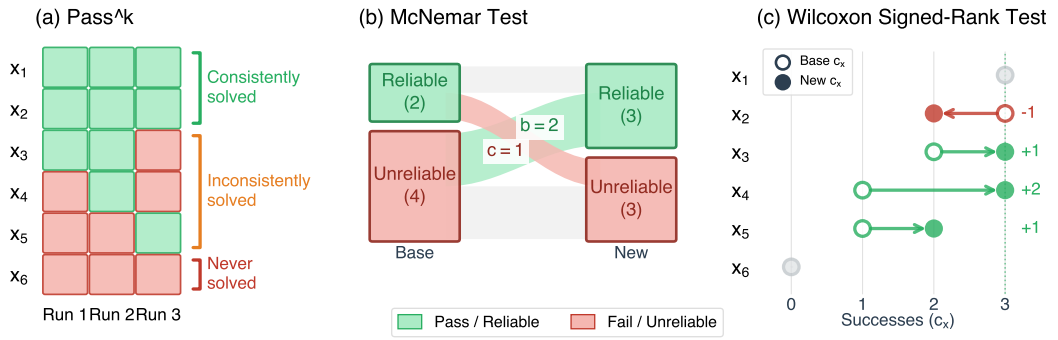


Figure 2: We illustrate three metrics for analyzing consistency in agent performance over multiple runs of the same task. (a) Pass^k (repeated-run success) estimates the probability that k executions of a task succeed, averaged across tasks. (b) McNemar measures improvements and regressions in reliability between two settings by counting tasks that transition between being consistently solved and not. (c) Wilcoxon signed-rank test compares per-task success counts across settings, capturing incremental changes in consistency even when full reliability is not achieved.

models. We primarily consider strong frontier models, including GPT-5 (Singh et al., 2025), Claude Sonnet 4.6 (Anthropic, 2026), and Kimi 2.5 (Bai et al., 2026), to assess whether reliability challenges persist in high-performing systems. We additionally include smaller and open-source models, such as Qwen-3VL-8B-Instruct (Bai et al., 2025), OpenCUA (Wang et al., 2025), and UI-TARS-1.5-7B (Qin et al., 2025), to enable controlled experiments and examine how reliability varies with model capability. For models with provided execution setups, we use the default running scripts in OSWorld (Xie et al., 2024). Otherwise, we adopt the Agent S3 (Gonzalez-Pumariega et al., 2026) settings for action space and grounding and add an (S3) suffix. All models use API required temperature (e.g. temperature 1 for GPT-5) or temperature 0.7 unless otherwise specified.

3 Stochasticity is Not the Bottleneck

If randomness in token sampling or environment dynamics drove unreliability, enforcing determinism would restore it, and the rest of our analysis would be unnecessary. We probe this directly with two interventions: *agent-side determinism* (temperature-0 batch-invariant decoding plus a fixed high-level plan reused across runs) and *controlled environment perturbations* (non-functional variations in observations). Across both, stochasticity is not the dominant axis of unreliability, and suppressing it can in fact *degrade* reliability.

3.1 Deterministic Agent Execution

Replacing stochastic decoding with temperature-0 batch-invariant inference yields a model-dependent and diagnostic result. OpenCUA and UI-TARS-1.5 improve significantly ($b - c = 20$ and 19), consistent with the textbook expectation. *Qwen, however, regresses by 20 tasks* ($b - c = -20$). Three models, three different reactions to the same intervention, so removing sampling noise is not a universal fix: for at least one model, sampling diversity was actively protecting reliability, likely by enabling recovery from a deterministic decoding path that happens to be locally suboptimal. Across all three, Pass^3 remains well below Pass^1: even where determinism helps, it does not close the gap.

We next add strategy determinism: the agent is constrained to follow a single high-level plan reused across runs (Appendix C.1.1, C.1.2), pinning down both token-level and strategy-level variation and leaving only environment dynamics. This partially mitigates Qwen's regression ($b - c = -1$, no longer significant) and preserves the gains for OpenCUA ($b{-}c{=}23$) and UI-TARS-1.5 ($b - c = 14$). Yet none of the models gain meaningful reliability over the stochastic baseline. Suppressing both token-level and strategy-level variability does not approach perfect Pass^3: the residual unreliability must originate elsewhere.

3.2 Sensitivity to Environment Noise

We perturb the environment along non-functional axes: cosmetic differences that do not change task correctness but alter the observation distribution (Appendix F). Even small cosmetic changes degrade reliability in the strongest models: GPT-5 regresses by 10 tasks ($b - c = -10$) and Claude by 20 ($b - c = -20$, $\Delta c_x = -0.080$). Kimi is roughly unaffected, but starts from a low Pass^3 of

Model	Setting	Pass^1	Pass^3	$b - c$	Δc_x
Qwen (S3)	Baseline	0.329	0.222	–	–
Qwen (S3)	Deterministic	0.293	0.166	-20*	-0.108*
Qwen (S3)	Strategy Determinism	0.313	0.219	-1	-0.047
OpenCUA	Baseline	0.226	0.125	–	–
OpenCUA	Deterministic	0.259	0.180	20*	0.097*
OpenCUA	Strategy Determinism	0.262	0.188	23*	0.108*
UI-TARS-1.5	Baseline	0.253	0.152	–	–
UI-TARS-1.5	Deterministic	0.290	0.205	19*	0.111*
UI-TARS-1.5	Strategy Determinism	0.264	0.169	14*	0

Table 1: Reliability metrics across deterministic decoding strategies. McNemar and Wilcoxon statistics are computed relative to the stochastic Baseline setting. An asterisk (*) denotes statistical significance at $p < 0.05$ under either test.

Model	Condition	Pass^1	Pass^3	$b - c$	Δc_x
GPT-5 (S3)	Baseline	0.576	0.454	–	–
GPT-5 (S3)	Perturbed	0.561	0.416	-10	-0.033
Claude	Baseline	0.702	0.612	–	–
Claude	Perturbed	0.675	0.557	-20*	-0.080*
Kimi	Baseline	0.508	0.357	–	–
Kimi	Perturbed	0.516	0.355	-1	0.025

Table 2: Reliability metrics under environment perturbation. McNemar and Wilcoxon statistics are computed relative to the fixed environment Baseline setting. An asterisk (*) denotes statistical significance at $p < 0.05$ under either test.

0.357, so there is little reliability left to lose.

The Claude regression is the most informative: it is the strongest model on the unperturbed setting (Pass^3 = 0.612) yet the most sensitive to cosmetic change. Capability and robustness to environment shift do not co-vary, and a model that solves more tasks in a fixed environment is not necessarily one that solves them *reliably* when the environment shifts: the worst-case combination for deployment.

Taken together. Removing sampling noise does not consistently help, and the strongest agents the strongest agent in our suite (Claude, Pass^3 = 0.612) is the most fragile to environmental perturbation. If sampling and environment noise were the dominant sources of unreliability, deterministic execution in an unperturbed environment should approach perfect Pass^k, but it does not. The remaining gap must come from the task and the agent itself, which we turn to next.

4 Ambiguity is Irreducible: Interaction is the Lever

Stochasticity is not the main culprit (§3.1), so we next ask whether the gap originates in the task. OSWorld instructions are written for human readers and admit multiple interpretations that the evaluator does not equally accept; an agent sampling among them lands on the evaluator-aligned one only some of the time. Static rewrites help, but the durable fix is to let the agent resolve ambiguity *interactively*, which produces the largest single intervention effect in the paper.

4.1 Clarification Before Execution

As a first test, we rewrite each task instruction to expose the success criteria the evaluator actually checks, drawing on the evaluation script and its helpers but excluding hints that would trivialize the task (Appendix E). Clarification yields statistically significant gains across every frontier model. GPT-5 sees its largest single-intervention gain of the paper (Pass^3: $0.454 \rightarrow 0.576$, $b - c = 48$, Δc_x=0.327); Claude moves from 0.612 to 0.673 ($b - c = 22$); Kimi shows a significant Wilcoxon shift ($\Delta c_x = 0.141$) indicating consistent partial progress, even where Pass^3 transitions are smaller. A one-time rewrite of instructions recovers more reliability than the strongest determinism intervention, suggesting that a significant share of CUA unreliability is not the agent's fault, but a specification problem the evaluation methodology has been silently tolerating.

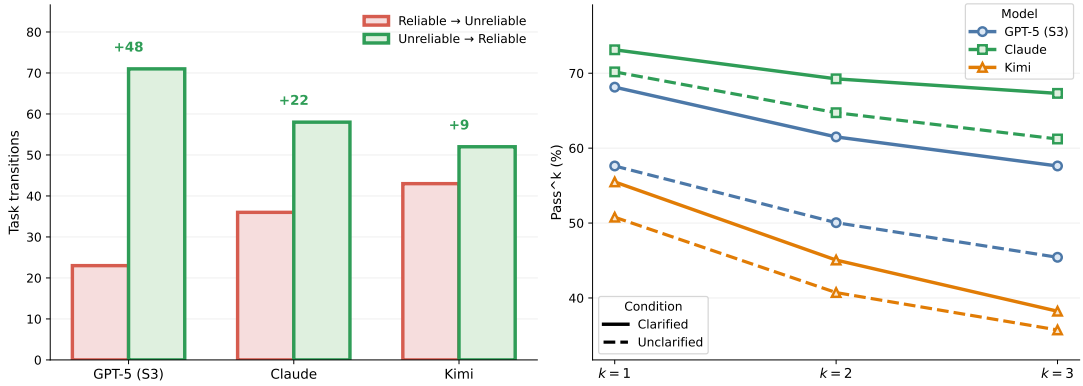


Figure 3: Task-level transitions in reliability under instruction clarification measured using McNemar analysis (left), and repeated-run success under clarified and unclarified instructions measured using Pass^k (right).

4.2 Clarification During User Interaction

Static clarification is bounded: a rewrite cannot anticipate ambiguities that surface only in a particular agent's trajectory. We therefore introduce a user simulator that observes the agent's failed trajectory and produces targeted feedback before the next attempt (Appendix C.3). To isolate *feedback content* from *additional attempts*, we compare Retry (Clarify) against a Retry (Binary) baseline given the same retry budget but only success/failure signals.

The results are the strongest in the paper (Table 3). Retry (Clarify) dominates every alternative on every model and every metric. For Kimi, Pass^3 rises from 0.357 to 0.634: a **+28 pp improvement and a near-doubling of reliability**, with 100 more tasks reliably solved than not ($b - c = 100$). GPT-5 reaches Pass^3 = 0.632 and Claude 0.706, both records for their rows. Retrying *with* feedback substantially outperforms retrying *without*, isolating the value of feedback content from the value of additional attempts.

The interaction thesis. (1) Static clarification helps but is bounded by what an author can anticipate. (2) Blind retries help less than feedback-driven retries on the same budget, so the gains are not just from more attempts. (3) The largest single intervention in the paper is targeted, on-the-fly clarification. Ambiguity is irreducible at specification time, and a reliable computer-use agent will be one that treats clarification as a first-class capability. Benchmarks that score agents purely on autonomous one-shot execution may be measuring a setting no real deployment will resemble.

5 Planning Variability: When Models Disagree About What Helps

Even with clarified instructions, agents still fail inconsistently on the same task. Stochasticity (§3.1) and ambiguity (§4.1) do not exhaust the gap; the residual source is the agent itself, selecting different strategies across runs with varying robustness. We test this by feeding evidence from prior runs into subsequent runs: extracting structured feedback into a reusable plan (Iteration 1) and iteratively refining that plan (Iteration 2). Unlike strategy determinism (§3.1), plan extraction derives the plan from observed behavior, not from the initial screenshot alone.

5.1 Effect of Plan Extraction

For Iteration 1, we run multiple rollouts per task with clarified instructions, extract structured feedback over successful behaviors and recurring failures (Appendix C.4.1), and synthesize a plan to guide the next run. Ground-truth success labels gate the feedback to avoid confounds from judge quality: if all rollouts succeed we skip feedback; if all fail, we extract from partial successes to encourage exploration (Appendix C.4.4). We modify the Behavior Judge of Gonzalez-Pumariega et al. (2026) to operate over behavior-narrative representations.

The headline result is that models disagree. GPT-5 and Kimi benefit (GPT-5: $+2.9$ pp Pass^1, $+4.2$ pp Pass^3, $\Delta c_x = 0.086$ Wilcoxon; Kimi: $+5.3$ pp Pass^3, $b - c = 19$), but *Claude regresses* significantly ($b - c = -15$, $\Delta c_x = -0.055$). Plan extraction is not universal: it actively harms one of three frontier models we evaluate.

Model	Setting	Pass^1	Pass^3	$b - c$	Δc_x
GPT-5 (S3)	Original	0.576	0.454	–	–
GPT-5 (S3)	Clarified	0.681	0.576	48*	0.327*
GPT-5 (S3)	Retry (Binary)	0.645	0.557	41*	0.216*
GPT-5 (S3)	Retry (Clarify)	0.730	0.632	68*	0.474*
Claude	Original	0.702	0.612	–	–
Claude	Clarified	0.731	0.673	22*	0.089
Claude	Retry (Binary)	0.732	0.668	20*	0.091
Claude	Retry (Clarify)	0.774	0.706	34*	0.216*
Kimi	Original	0.508	0.357	–	–
Kimi	Clarified	0.555	0.382	9	0.141*
Kimi	Retry (Binary)	0.623	0.496	50*	0.346*
Kimi	Retry (Clarify)	0.727	0.634	100*	0.657*

Table 3: Reliability metrics across instruction ambiguity interventions. McNemar and Wilcoxon statistics are computed relative to the Original baseline setting. An asterisk (*) denotes statistical significance at $p < 0.05$ under either test.

Model	Iteration	Pass^1	Pass^3	$b - c$	Δc_x
GPT-5 (S3)	Iteration 0	0.681	0.576	–	–
GPT-5 (S3)	Iteration 1	0.710	0.618	15	0.086*
GPT-5 (S3)	Iteration 2	0.725	0.651	27*	0.130*
Claude	Iteration 0	0.731	0.673	–	–
Claude	Iteration 1	0.713	0.632	-15*	-0.055*
Claude	Iteration 2	0.730	0.662	-4	-0.006
Kimi	Iteration 0	0.555	0.382	–	–
Kimi	Iteration 1	0.567	0.435	19*	0.036
Kimi	Iteration 2	0.563	0.424	15	0.025

Table 4: Reliability metrics across iterations of plan extraction and iterative plan refinement. McNemar and Wilcoxon statistics are computed relative to Iteration 0. An asterisk (*) denotes statistical significance at $p < 0.05$ under either test.

Why does Claude regress? Trajectory inspection suggests Claude has a strong prior toward *code-based solutions*: scripting tasks via shell or library commands rather than driving the GUI. Code paths succeed quickly and silently, mutating state in ways the Behavior Judge (which observes a GUI-behavior narrative) struggles to summarize. Feedback extracted from such trajectories is incomplete or misaligned, and conditioning the next run on it can lock the agent into a less reliable strategy than the one it would have chosen unguided. The same intervention that helps GUI-first models hurts a code-first model: **interventions interact with the agent's prior, not just with the task**.

5.2 Effect of Iterative Plan Refinement

For Iteration 2, we run the agent again under the Iteration 1 plan, collect new feedback, and update the plan with the same extraction prompt (Appendix C.4.2). If all Iteration 2 rollouts fail, we draw on successful rollouts from earlier iterations (Appendix C.4.3).

GPT-5 continues to improve (Pass^3: $0.576 \rightarrow 0.651$ across Iterations 0–2, $b - c = 27$, $\Delta c_x = 0.130$, both significant). Kimi regresses slightly from Iteration 1 but stays above Iteration 0. Claude's regression no longer reaches significance ($b - c = -4$, $\Delta c_x = 0.006$): successive refinement partially *repairs* the bad initial extraction without flipping it to a net gain.

Patterns by model. Across our interventions, three patterns emerge in this evaluation:

- **GPT-5** responds positively to nearly every intervention we test, gaining from both static clarification and iterative plan refinement.
- **Kimi** gains the most from interactive clarification (+28 pp Pass^3) and shows positive but smaller effects from plan extraction, while starting from the lowest baseline.
- **Claude** is the strongest unguided model in our suite but is the most sensitive to non-functional environment perturbation.

Compounding gains. For GPT-5, compounding clarified instructions (§4.1), interactive feedback (§4.2), and refined plans (§5.2) lifts Pass^3 from 0.454 to 0.651, closing roughly half the original reliability gap relative to the Pass^1 ceiling. The remaining half is not addressed by anything in our intervention space and is, we believe, where future work must focus.

6 Discussion and Conclusion

Computer-use agents exhibit a large reliability gap: more than half of the apparent capability of a strong agent disappears once we ask for reproducibility. The natural fix, enforcing determinism, does not work, and can backfire. The interventions that *do* work share a common shape: they treat ambiguity and strategy variability as the real bottlenecks, and they handle them dynamically rather than statically.

Stochasticity is not the dominant axis. Deterministic decoding helps some models (OpenCUA, UI-TARS-1.5) and *hurts* others (Qwen, $b - c = -20$). Adding strategy determinism stabilizes execution without exceeding the stochastic baseline. Worse, the strongest agent (Claude) is the most fragile to cosmetic environment perturbations ($b - c = -20$). Removing stochasticity strips the agent of the variability it uses to recover from suboptimal initial paths, so brittleness shifts from across-run variance to within-run rigidity. A more promising direction combines symbolic structure with stochastic decoding, using symbolic representations to guide execution while preserving local adaptivity (Gao et al., 2023; Wang et al., 2024b; Chen et al., 2023).

Ambiguity is irreducible; interaction is the lever. Clarifying task specifications produces the largest static gain in the paper (GPT-5: $b - c = 48$, $\Delta c_x = 0.327$). But the largest gain of any kind comes from *interactive* clarification, where a user simulator provides targeted feedback on failed runs: Kimi gains $+28$ pp on Pass^3. A binary retry baseline with the same budget recovers far less, so the value lies in the feedback, not the additional attempts. Real-world tasks are never free of ambiguity at specification time, so the durable fix is not better instructions but agents that resolve ambiguity through interaction (Wang et al., 2024a; Handa et al., 2024; Piriyakulkij et al., 2024).

Interventions interact with the agent's prior. Plan extraction helps GPT-5 and Kimi ($b - c = 15, 19$) but regresses Claude ($b - c = -15$); we attribute this to Claude's code-first solution style interacting poorly with our GUI-narrative behavior judge (§5.1). The general lesson is that reliability interventions are not portable across agents with different behavioral priors. Iterative refinement (Shinn et al., 2023; Madaan et al., 2023) and in-context demonstrations from prior trajectories (Gupta et al., 2025) are promising primitives, but composing them robustly requires explicit accounting for solution style.

An evaluation protocol the community can adopt. We summarize the methodological piece as a reusable recipe:

1. **Report Pass^k alongside Pass@k**: the gap between them is the reliability gap; Pass@k alone overstates capability two-fold or more.
2. **Use $n \geq 3$ repeated executions per task** (we use n=3 for most experiments, n=10 where compute allows).
3. **Compare settings with paired tests over per-task outcomes** (McNemar for transitions, Wilcoxon signed-rank for count shifts). Aggregate Pass^k differences hide direction.

How far from deployment-grade reliability? Aerospace, automotive, medical, and regulated AI systems demand reliability targets of multiple nines. The best Pass^10 figure in our experiments is $\sim$36%. After compounding clarified instructions, interactive feedback, and refined plans, our strongest configuration reaches Pass^3 ≈ 0.65 for GPT-5: roughly half the gap closed. Computer-use agents remain an order of magnitude below safety-critical bars; closing the rest will require mechanisms aimed at consistency, not capability.

7 Limitations

All experiments use OSWorld (Xie et al., 2024), so whether the patterns we report transfer to web navigation or enterprise software remains open. Per-model patterns are derived from six agents and may not generalize (for example, Claude's code-first regression under plan extraction). We use n=3 repeated executions for most experiments which limits statistical power for small per-task effects and is why we report $b-c$ and Δc_x alongside p-value significance. Plan extraction relies on a GUI-narrative Behavior Judge that under-describes code-based trajectories (§5.1), and the user simulator is an LLM rather than a real human.

References

Jiaxin Ai, Yukang Feng, Fanrui Zhang, Jianwen Sun, Zizhen Li, Chuanhao Li, Yifan Chang, Wenxiao Wu, Ruoxi Wang, Mingliang Zhai, and Kaipeng Zhang. 2025. Prosoftarena: Benchmarking hierarchical capabilities of multimodal agents in professional software environments. *Preprint*, arXiv:2601.02399.

AI HLEG. 2019. Ethics guidelines for trustworthy ai.

Anthropic. 2026. Claude sonnet 4.6 system card. https://www-cdn.anthropic.com/bbd8ef16d70b7a1665f14f306ee88b53f686aa75.pdf.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, Yanru Chen, and 306 others. 2026. Kimi k2.5: Visual agentic intelligence. *Preprint*, arXiv:2602.02276.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Preprint*, arXiv:2211.12588.

Yutong Dai, Krithika Ramakrishnan, Jing Gu, Matthew Fernandez, Yanqi Luo, Viraj Prabhu, Zhenyu Hu, Silvio Savarese, Caiming Xiong, Zeyuan Chen, and Ran Xu. 2025. Scuba: Salesforce computer use benchmark. *Preprint*, arXiv:2509.26506.

Xiang Deng and 1 others. 2023. Mind2web: Towards a generalist agent for the web. In *NeurIPS*.

Eaton Corporation. 2019. Introduction to functional safety and iec 61508. Technical report, Eaton.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. *Preprint*, arXiv:2211.10435.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. 2026. Scaling agents for computer use. *Preprint*, arXiv:2510.02250.

Shivanshu Gupta, Sameer Singh, Ashish Sabharwal, Tushar Khot, and Ben Bogin. 2025. Leveraging in-context learning for language model agents. *Preprint*, arXiv:2506.13109.

Kunal Handa, Yarin Gal, Ellie Pavlick, Noah Goodman, Jacob Andreas, Alex Tamkin, and Belinda Z. Li. 2024. Bayesian preference elicitation with language models. *Preprint*, arXiv:2403.05534.

Intuition Labs. 2022. Iec 62304: Medical device software life cycle processes. Technical report, Intuition Labs.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Parasoft Corporation. 2023. Do-178c software compliance for aerospace & defense. Technical report, Parasoft Corporation.

Wasu Top Piriyakulkij, Volodymyr Kuleshov, and Kevin Ellis. 2024. Active preference inference using language models and probabilistic reasoning. *Preprint*, arXiv:2312.12009.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *Preprint*, arXiv:2501.12326.

ROHM Semiconductor. 2020. Iso 26262 functional safety standard for automotive applications. Technical report, ROHM Co., Ltd.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 465

others. 2025. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Elham Tabassi. 2023. *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*.

Huaxiaoyue Wang, Nathaniel Chin, Gonzalo Gonzalez-Pumariega, Xiangwan Sun, Neha Sunkara, Maximus Adrian Pace, Jeannette Bohg, and Sanjiban Choudhury. 2024a. Apricot: Active preference learning and constraint-aware task planning with llms. *Preprint*, arXiv:2410.19656.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better llm agents. *Preprint*, arXiv:2402.01030.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, and 23 others. 2025. Opencua: Open foundations for computer-use agents. *Preprint*, arXiv:2508.09123.

Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

Jing Wu, Daphne Barretto, Yiye Chen, Nicholas Gydé, Yanan Jian, Yuhang He, and Vibhav Vineet. 2026. Os-marathon: Benchmarking computer-use agents on long-horizon repetitive tasks. *Preprint*, arXiv:2601.20650.

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

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

Henry Hengyuan Zhao, Kaiming Yang, Wendi Yu, Difei Gao, and Mike Zheng Shou. 2026. Worldgui: An interactive benchmark for desktop gui automation from any starting point. *Preprint*, arXiv:2502.08047.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and 1 others. 2024. Webarena: A realistic web environment for building autonomous agents. *ICLR*.

A Ethical Consideration

Because computer-use agents act on real systems, the reliability gap we document has direct safety implications: an unreliable agent is not merely less useful but can be actively harmful (e.g., overwriting the wrong file), and current Pass^k figures remain far below the multiple-nines targets safety-critical domains demand. Our experiments use OSWorld (Xie et al., 2024), which contains synthetic and public-content tasks and no personal data; no human subjects were involved, so no IRB approval was required. We also used LLMs to assist with prose, figure code, and prompt drafting, but not to author research claims or analyze results.

B Related Work

Computer-Use Agents and Benchmarks. A central goal of recent work is to build agents that can execute real-world tasks by interacting directly with computing environments. OSWorld provides a comprehensive setting for general computer use, requiring agents to operate across applications, file systems, and web environments (Xie et al., 2024), while a growing body of work studies related settings such as robust and long-horizon GUI interaction (Zhao et al., 2026; Wu et al., 2026), web navigation (Zhou et al., 2024; Koh et al., 2024; Deng et al., 2023), and enterprise software (Ai et al., 2025; Dai et al., 2025). Despite differences in environment and task design, these benchmarks primarily report single-run performance metrics such as success rate or task completion, rather than measuring consistency across repeated executions. In contrast, we focus on reliability as a primary evaluation target, measuring whether agents can consistently succeed on the same task across multiple runs.

Measuring Reliability in Agents. Evaluation of agents is commonly based on sampling-based metrics that estimate success across multiple attempts, with Pass@k being one of the most widely used metrics for large language models. Pass@k measures whether an agent can produce at least one successful outcome across k independent samples, and is often interpreted as an upper bound on task success under repeated sampling (Chen et al., 2021). However, this metric does not capture whether an agent can succeed consistently, as it only requires success in a single attempt. More recently, τ-bench (Yao et al., 2024) introduces Pass^k as a reliability metric, defined as the probability that an agent succeeds across all k repeated executions of the same task, which can be viewed as a lower bound on consistent performance. While Pass^k captures consistency, it remains an aggregate metric and does not

capture how reliability changes at the level of individual tasks across settings. In this work, we build upon this metric with paired statistical tests over repeated executions, allowing us to detect per-task improvements and regressions in consistency and to compare reliability changes across conditions.

C System Prompts

C.1 Strategy Determinism

C.1.1 Sampling Plan Prompt

Listing 1: Strategy Determinism Plan Sampling

```
You are an expert at creating step-by-step plans
     for GUI automation tasks.

Task: {instruction}

Please analyze the current screenshot and create
     a detailed step-by-step plan to accomplish
    this task.

Your plan should:
1. Be specific about what to click, type, or
    interact with
2. Include the sequence of actions needed
3. Be detailed enough that another agent could
    follow it

Respond with a numbered list of steps:
1. [Action description]
2. [Action description]
...

Only provide the plan - do not execute any
    actions.
```

C.1.2 Reusing Plan Prompt

Listing 2: Strategy Determinism Plan Addon

```
[PLAN TO FOLLOW] PRE-GENERATED STEP-BY-STEP PLAN:
     Below is a detailed plan for completing the
     task: `{instruction}`
You must follow this plan step-by-step to
    succeed at the task.

{plan_text}

IMPORTANT: Follow this plan closely. Each step
    should guide your next action. Adapt the
    plan to the current screen state when
    necessary, but stay true to the overall plan
     structure.
```

C.2 Clarification Before Execution

C.2.1 Instruction Clarification Prompt

Listing 3: Instruction Clarification Prompt

```
You are improving the clarity of an OSWorld
    evaluation task instruction. The current
    instruction is vague and doesn't provide
    enough detail for an AI agent to know
    exactly what to do based on the evaluation
    criteria.

The goal is to make MINIMAL clarifications while
     preserving the natural, human-like tone. Do
     NOT rewrite as step-by-step instructions.
    Only clarify ambiguous details that are
    checked by the evaluator but missing from
    the instruction.

## Current Task Configuration:
```json
{example_config}
```

Evaluator Function Implementation(s):
```python
{func_implementations}
```

Task:

Analyze the instruction: {instruction}

Identify what the evaluator checks that isn't
    clear from the instruction, then provide a
    minimally clarified version that:
1. **Keeps the natural human tone** - should
    sound like a human said it
2. **Only adds missing details** that the
    evaluator checks for
3. **Specifies exact file names/formats/
    locations** if the evaluator expects them
4. **Does NOT become a step-by-step procedure** -
     stay conversational

Additional criteria:
1. The clarified instruction will replace the
    original instruction - make sure to retain
    essential details to avoid losing current
    context (e.g. do not omit links).
2. For tasks that have exact file checks, be
    very careful in accidentally adding or
    preventing information to the instruction
    that would cause it to no longer match the
    evaluator's criteria.

Format your response as:
<thoughts>
[Think through what's ambiguous and what the
    evaluator actually checks for]
</thoughts>

<answer>
[The minimally clarified instruction in natural
    language]
</answer>
```

C.2.2 Clarification Failure Analysis Prompt

Listing 4: Clarification Failure Analysis Prompt

```
You are analyzing how instruction clarifications
     affect agent success rate on GUI automation
     tasks.

You will be given:
```

```
1. ORIGINAL (before) instruction and 3
     trajectory runs
2. CLARIFIED (after) instruction and 3
     trajectory runs
3. Success rate scores for all runs
4. **EVALUATOR FUNCTION**: The exact code that
     determines task success/failure

Your goal is to determine:
- How agent behavior changed between the
     original and clarified instructions
- Whether the instruction clarification had a
     meaningful impact on success rate
- What specific aspect of the clarification
     caused behavior changes

Each trajectory contains:
- **Visual Changes**: Fact captions describing
     GUI state changes at each step
- **Agent Reasoning**: The agent's internal
     thoughts and decision-making process

CRITICAL EVALUATION CONTEXT:

**Understanding the Evaluator**: The evaluator
     function shows EXACTLY what determines
     success/failure. Pay close attention to:
- **Exact matching requirements**: File names,
     paths, content, button states
- **Precise conditions**: Specific UI elements,
     file formats, directory structures
- **Failure points**: What tiny details can
     break the evaluation

Many evaluators use EXACT MATCHING, where small
     differences cause complete failure:
- Wrong button clicked vs right button
- File saved in wrong directory
- Missing file or incorrect file name
- Different file format or content
- UI element not in expected state

ANALYSIS APPROACH:

1. **Understanding Original Success Rate**
   - Analyze the 3 'before' runs: what did
       successful runs do RIGHT according to the
        evaluator?
   - For failures: what specific evaluator
       requirement did they miss?

2. **Understanding Clarified Success Rate**
   - Analyze the 3 'after' runs: what changed in
        agent behavior?
   - Do the new behaviors align with or conflict
        with evaluator requirements?

3. **Instruction Impact on Evaluator Alignment**
   - Did the clarification help agents meet
       evaluator requirements more precisely?
   - Did the clarification inadvertently steer
       agents away from evaluator requirements?
   - Are success rate changes due to better/
       worse evaluator alignment or just
       variance?

4. **Flag Assignment**
   Choose ONE flag that best describes this case:

   **Instructions Need Manual Correction:**
   - IMPOSSIBLE_TASK: Clarification made the task
        impossible to complete
   - TOO_TRIVIAL: Clarification gave away the
       solution, making task too easy
   - HARMFUL_CONSTRAINTS: Added unnecessary
       constraints that hurt success rate
   - REMOVED_HELPFUL_AMBIGUITY: Removed ambiguity
        the agent was successfully leveraging
   - OTHER_MISINTERPRETATION: Other instruction
       problem (rarely used)

   **Clarification Effect Analysis:**
   - GENUINE_IMPROVEMENT: Instruction change
       demonstrably helped agents meet evaluator
        requirements more consistently
   - RANDOM_VARIANCE: Success rate change
       appears unrelated to instruction, likely
       variance

   **Important Notes:**
   - Focus on evaluator alignment: did the
       clarification help or hurt the agent's
       ability to satisfy the exact evaluator
       requirements?
   - Small execution differences can completely
       break exact matching evaluators

   When choosing a flag, consider the success
       rates and evaluator requirements:
   - It does not make sense to assign
       GENUINE_IMPROVEMENT if the success rate
       did not improve
   - It does not make sense to assign
       IMPOSSIBLE_TASK if the success rate
       improved
   - If success rate decreased significantly,
       look for what evaluator requirement the
       clarification broke
   - If success rate improved, verify the
       clarification actually helped meet
       evaluator criteria (not just luck)

OUTPUT FORMAT:

<thinking>
[First understand the evaluator requirements
     thoroughly. Then analyze before/after
     trajectories systematically to see how agent
      behaviors changed relative to these
     requirements. Determine if success rate
     differences are due to better/worse
     evaluator alignment or variance.]
</thinking>

<answer>
Flag: [ONE of the flags above]
Analysis: [Concise explanation focusing on how
     the instruction change affected the agent's
     ability to meet the specific evaluator
     requirements. Explain what evaluator
     criteria were better/worse satisfied and why
      this led to the success rate change.]
</answer>
```

C.2.3 Input for Analysis Prompt

Listing 5: Clarification Failure Analysis Input

```
Task ID: {task_id}

ORIGINAL INSTRUCTION:
{before_instruction}

CLARIFIED INSTRUCTION:
{after_instruction}

EVALUATOR FUNCTION:
```python
{evaluator_implementation}

SUCCESS RATE COMPARISON:
ORIGINAL: {before_successes}/{before_total}
    successes ({before_successes/before_total
    :.1%})
CLARIFIED: {after_successes}/{after_total}
    successes ({after_successes/after_total
    :.1%})

=== ORIGINAL INSTRUCTION TRAJECTORIES ===

--- Run {run_index} ({SUCCESS|FAILURE}, Score: {
    score}) ---
Visual Changes:
{fact_captions...}

Agent Reasoning:
{trajectory_reasoning...}

[repeated for each of 3 'before' runs]

=== CLARIFIED INSTRUCTION TRAJECTORIES ===

[same format for each of 3 'after' runs]

Please analyze how the instruction clarification
     affected agent behavior and assign the
    appropriate flag.
```

C.3 User Simulator

C.3.1 User Simulator: Failure Feedback Prompt

Listing 6: User Simulator Failure Feedback Prompt (Clarify Upon Retry)

```
You are the user who issued the following task
    to a computer-use agent:

"{instruction}"

The agent attempted the task but FAILED. You
    need to give the agent concise, actionable
    feedback about what went wrong so it can try
     again.

Below is your internal knowledge about the task:

{context}

{file_diff_section}

RULES FOR FEEDBACK:
1. Be specific about what is wrong -- mention
    exact values, file names, or formats that
    are incorrect.
2. Do NOT give step-by-step instructions. Just
    point out the problem.
3. Do NOT reveal evaluator function names or
    code.
4. Keep it to 2-4 sentences.
5. If you can identify the specific mismatch
    between what the agent produced and what you
     expected, state it clearly.
```

C.3.2 User Simulator: Context

Listing 7: User Simulator Context

```
The {context} variable contains:

## Task Configuration
```json
{full task JSON config including instruction,
    evaluator spec, setup config}
```

Evaluator Function
```python
{evaluator function source code found by AST-
    walking desktop_env/evaluators/}
```

Input/Expected Files
{file contents: images as base64 image_url
    blocks, text files in full, spreadsheets as
    CSV, files >10MB as metadata only}

Agent's trajectory
{step-by-step actions with screenshots from
    agent.executor.screenshot_inputs (last 40)}
```

### C.3.3 Binary Retry Signal

Listing 8: Binary Retry Signal (injected after failed evaluation)

```
Your previous attempt did not succeed. The task
    is not complete. Please try again.
```

C.3.4 Clarify Upon Retry Signal

Listing 9: Clarify Upon Retry Signal (injected after failed evaluation)

```
Your previous attempt did not succeed. Here is
    feedback from the user:
{feedback from user simulator}
```

C.4 Plan Extraction and Refinement

C.4.1 Plan Extraction and Refinement Base Prompt

Listing 10: Plan Extraction and Refinement Prompt.

```
You are analyzing agent trajectories to provide
    reflexion feedback for improving future
    performance.

You are given multiple rollouts from the SAME
    policy on the SAME task:
```
```

```
- Some SUCCESSFUL rollouts (score > 0)
- Some FAILED rollouts (score = 0)

{PREVIOUS_FEEDBACK_SECTION}

For each rollout, you will receive:
1. **Visual Changes**: Fact captions describing
     what changed between screenshots at each
     step
2. **Agent Reasoning**: The agent's internal
     thoughts and decision-making process at each
      step

Your goal is to identify what worked in
     successful runs versus what went wrong in
     failed runs, then provide actionable
     feedback for future attempts.

ANALYSIS APPROACH:

1. **Compare Trajectories Step-by-Step**
   - Compare both visual changes and agent
       reasoning between successful/failed runs
   - Identify where successful and failed runs
       diverge in actions AND thinking
   - Note different approaches taken by
       successful vs failed runs
   - Look for consistent patterns across
       successes and failures

2. **Key Success Factors**
   - What specific actions or strategies led to
       success?
   - What reasoning patterns did successful runs
        exhibit?
   - Were there critical steps that successful
       runs handled correctly?
   - What environmental awareness did successful
        runs demonstrate?

3. **Failure Analysis**
   - What mistakes did failed runs make in
       actions or reasoning?
   - Were there flawed reasoning patterns or
       incorrect decision making?
   - Did failed runs misunderstand the task or
       environment?
   - Were there missed opportunities or
       incorrect decisions?

4. **Actionable Insights**
   - What should the agent think about
       differently next time?
   - What should the agent do differently next
       time?
   - What should the agent avoid based on failed
        attempts?
   - Are there general principles or heuristics
       that emerge?

OUTPUT FORMAT:

<analysis>
[Detailed step-by-step comparison of successful
     vs failed trajectories, analyzing both
     visual changes and agent reasoning,
     identifying key divergence points and
     patterns]
</analysis>

<feedback>
DO:
1. [Specific actionable instruction based on
     successful patterns]
2. [Another positive action or strategy to
     follow]
3. [Key behavior or approach to adopt]
...
(up to a max of 10 items and can stop earlier if
      not applicable)

DON'T:
1. [Specific mistake or behavior to avoid from
     failed runs]
2. [Common error pattern to prevent]
3. [Action or thinking pattern that leads to
     failure]
...
(up to a max of 10 items and can stop earlier if
      not applicable)

PLAN:
[Step-by-step execution plan extracted from
     successful runs - concrete sequence of
     actions that leads to success]
</feedback>
```

C.4.2 Plan Refinement and Addon Prompt

Listing 11: Plan Refinement Addon Prompt (PREVIOUS_FEEDBACK_SECTION)

```
PREVIOUS FEEDBACK:
```
{previous_feedback}
```

ANALYSIS GOAL:
The previous feedback was applied, but the
     results show it may need refinement. You
     need to:
1. Identify which parts of the previous feedback
      were helpful vs harmful
2. Understand why the feedback led to the
     current outcomes
3. Refine the feedback based on new evidence
```

C.4.3 Plan Extraction with Historical Success

Listing 12: Plan Extraction with Historical Success

```
You are analyzing failed agent trajectories
     against a successful historical reference to
      generate corrective feedback.

SITUATION:
All current attempts failed, but we have a
     successful run from a previous trial to
     learn from.

HISTORICAL SUCCESSFUL RUN:
{historical_success_info}

CURRENT FAILED ATTEMPTS:
You will be shown multiple failed attempts from
     the current trial.

ANALYSIS GOAL:
```

```
Compare the failed attempts against the
     historical success to identify:
1. What the successful run did right that
     current attempts are missing
2. What systematic errors current attempts are
     making
3. How to guide future attempts toward the
     successful pattern

For each run, you will see:
1. **Visual Changes**: Fact captions describing
     what changed between screenshots at each
     step
2. **Agent Reasoning**: The agent's internal
     thoughts and decision-making process at each
      step

ANALYSIS APPROACH:

1. **Success Pattern Analysis**
   - What made the historical run successful?
   - What key decisions, actions, or reasoning
       patterns led to success?
   - What environmental awareness did it
       demonstrate?

2. **Current Failure Analysis**
   - How do current failures differ from the
       successful pattern?
   - What systematic mistakes are being repeated?

   - Where do current attempts go wrong compared
        to the success?

3. **Corrective Strategy**
   - How can future attempts align with the
       successful pattern?
   - What specific changes in reasoning or
       actions are needed?
   - What should be avoided based on current
       failures?

OUTPUT FORMAT:

<analysis>
[Detailed comparison of historical success vs
     current failures, identifying key
     differences and systematic error patterns]
</analysis>

<feedback>
DO:
1. [Key strategy from the successful run to
     adopt]
2. [Specific action or approach that led to
     success]
3. [Reasoning pattern that worked in the
     successful case]
...
(up to a max of 10 items and can stop earlier if
      not applicable)

DON'T:
1. [Specific mistake in current attempts to
     avoid]
2. [Error pattern that differs from successful
     approach]
3. [Action or thinking that leads to failure vs
     success]
...
(up to a max of 10 items and can stop earlier if
      not applicable)

PLAN:
[Step-by-step execution plan extracted from the
     historical successful run - concrete
     sequence of actions to replicate]
</feedback>
```

C.4.4 Plan Extraction from Failures

Listing 13: Plan Extraction from Failures

```
You are analyzing failed agent trajectories to
     generate diagnostic feedback when no
     successful examples exist.

SITUATION:
All attempts have failed, and there are no
     successful runs from previous trials to
     reference.

CURRENT FAILED ATTEMPTS:
You will be shown multiple failed attempts. Your
      goal is to identify the most promising
     approach and provide guidance to help future
      attempts succeed.

ANALYSIS GOAL:
Since all attempts failed, focus on:
1. Which failure showed the most promise/
     progress
2. What systematic errors are preventing success
3. What fundamental strategies might lead to
     success

For each failed run, you will see:
1. **Visual Changes**: Fact captions describing
     what changed between screenshots at each
     step
2. **Agent Reasoning**: The agent's internal
     thoughts and decision-making process at each
      step

ANALYSIS APPROACH:

1. **Comparative Failure Analysis**
   - Which attempt made the most progress before
        failing?
   - What different approaches were tried?
   - What common error patterns appear across
       attempts?

2. **Root Cause Identification**
   - What fundamental issues are causing
       failures?
   - Are failures due to misunderstanding, poor
       execution, or environment issues?
   - What capabilities seem to be missing?

3. **Diagnostic Strategy**
   - What should future attempts focus on first?
   - What basic principles or approaches might
       help?
   - What specific errors should be avoided?

OUTPUT FORMAT:

<analysis>
```

```
[Detailed analysis of failure patterns,
    identifying the most promising approach and
    systematic issues preventing success]
</analysis>

<feedback>
DO:
1. [Basic strategy that might lead to success]
2. [Fundamental approach to try based on most
    promising failure]
3. [Key capability or awareness to develop]
...
(up to a max of 10 items and can stop earlier if
     not applicable)

DON'T:
1. [Common error pattern to avoid across all
    attempts]
2. [Systematic mistake that prevents success]
3. [Flawed approach or reasoning to prevent]
...
(up to a max of 10 items and can stop earlier if
     not applicable)
</feedback>
```

C.4.5 Planning Feedback Addon

Listing 14: Planning Feedback Addon

```
[REFLECTION FEEDBACK] LEARNING FROM PREVIOUS
    ATTEMPTS:

Task: `{instruction}`

Based on analysis of previous attempts at this
    task, here's what you should know:

{feedback_text}

IMPORTANT: Use this feedback to guide your
    approach. Pay attention to what has worked
    and what has failed in similar situations.
    Adapt your strategy based on these insights
    while staying focused on completing the task
     successfully.
```

D Qualitative Analysis

D.1 Clarified Instruction Examples

We present qualitative examples illustrating the effects of instruction clarification. Each example shows the original instruction and task context, the LLM-generated clarified instruction, and (when applicable) a human-corrected version. We highlight both successful clarifications that improve alignment with the evaluator, as well as failure cases where clarification introduces unintended artifacts such as impossible constraints or trivial solutions.

Example 1: Improved evaluator alignment

Original Instruction: *Append one entry of AI researcher Yann LeCun from Google Scholar into an existing table researchers.xlsx.*

Clarification Context

- Original `researchers.xlsx` file with formatting
- Evaluation JSON with arguments to evaluator function `literal_match`
- Implementation of `literal_match` function
- Initial screenshot (google.com)

> Open `/home/user/Desktop/researchers.xlsx` and append a single new row at the end for AI researcher Yann LeCun, using his Google Scholar profile at `https://scholar.google.com/citations?user=WLN3QrAAAAAJ&hl=en`. Fill the same six columns in this order—Name, All citations, h-index, i10-index, Top Cited Work, PDF Link of Top Cited Work—using the "All" values for the three metrics; for Top Cited Work, use the title of the first listed publication on his profile, and for the PDF link, open that publication and copy the external PDF/full-text link shown on its details page. Save the file.

Analysis

> The clarification directly aligned the agent with the evaluator's exact-matching requirements:
>
> - **File specification:** Provides the exact file path and explicitly instructs appending a single new row, ensuring correct list length and ordering.
> - **Source grounding:** Specifies the exact Google Scholar profile URL, avoiding ambiguity and failure modes such as search errors, CAPTCHA, or login interruptions.
> - **Schema alignment:** Defines the six required columns and their order, and specifies use of the "All" metrics, preventing mismatches in format or subset selection.
> - **Deterministic extraction:** Instructs selecting the first listed publication and extracting the external PDF/full-text link from the article page, leading to consistent outputs.
> - **Completion requirement:** Explicitly instructs saving the file, ensuring the final state matches evaluator expectations.
>
> In the original instruction, agents frequently failed to append a row, produced incomplete or incorrect fields (e.g., wrong paper or link), or became stuck during web navigation—all of which result in failure under `literal_match`. After clarification, agent behavior becomes consistent with the evaluator's requirements, improving success from 0% to 100%.

Example 2: Instruction - evaluator mismatch

Original Instruction: *I am a Chinese citizen and I want to go to Macau to watch a concert recently, but I have not yet applied for a visa for Macau. I live in Futian District, Shenzhen City. I heard that Shenzhen currently has 24-hour self-service check-in machines. Please help me find the addresses of 5 24-hour self-service check-in machines in Futian District and save them in Chinese in this open word document.*

Clarification Context

- Original `AllLocations.docx` file
- Evaluation JSON containing potential match arguments to evaluator function `fuzzy_place_math`
- Implementation of `fuzzy_place_math` function (incorrectly implemented, shown below)

```
def fuzzy_place_math(result_file_path,
      rules) -> float:
    if result_file_path is None:
        return 0.
    # a list of possible answers
    expect = rules["expected"]

    doc = Document(result_file_path)
    words_list = []
    for para in doc.paragraphs:
        words_list.extend(
          para.text.split())

    fuzzy_score_list = []
    for word in words_list:
        max_score = 0
        for ans in expect:
         score = fuzz.ratio(word, ans)
                / 100
        max_score = max(max_score, score)
      fuzzy_score_list.append(max_score)
   ####################################
    BUG: requires exactly 3 addresses,
    otherwise automatic failure
   ####################################
    if len(fuzzy_score_list) != 3:
        return 0.

    return sum(fuzzy_score_list) / 3
```

- Initial screenshot (empty `AllLocations.docx` file)

Clarified Instruction (LLM)

> I'm a Chinese citizen planning to go to Macau for a concert and haven't applied for the visa yet. I live in Futian District, Shenzhen. I heard there are 24-hour self-service check-in machines in Shenzhen. Please help me find three (3) addresses of 24-hour self-service check-in machines located in Futian District and put them into the currently open document named `AllLocations.docx` on my Desktop. Please write the addresses in Simplified Chinese, with each address alone on its own line, and don't add any other text, numbering, bullets, or extra spaces before/after. The document should contain only these three lines. Save the changes to the same file.

Analysis

> The clarification changed the required output from "5 addresses" to exactly three addresses and added strict formatting constraints (only three lines, no extra text/numbering/bullets, no extraneous spaces, save to the same file). The evaluator demands exactly 3 whitespace-separated tokens in `AllLocations.docx`; any additional tokens cause an automatic fail. Under the original instruction, agents included a title and five items (often with spaces), producing more than three tokens and yielding 0/3 successes. After clarification, agents produced exactly three standalone address lines and removed extras, aligning with the evaluator's len==3 check and enabling high fuzzy matches to the expected answers, resulting in 3/3 successes. This is a clear, instruction-driven alignment improvement rather than random variance.

Example 3: Human-corrected impossible task after clarification

Original Instruction: *I've noticed that the image on the second slide is too dim. Can you please enhance its brightness for me? Save the adjusted image on the Desktop and name it "background.png". Thank you!*

Clarification Context

- Original `PPT-Template_widescreen.pptx` file with formatting
- Evaluation JSON with arguments to evaluator function `check_brightness_decrease_and_structure_sim`
- Implementation of `check_brightness_decrease_and_structure_sim` function
- Initial screenshot (opened `PPT-Template_widescreen.pptx` file)

Clarified Instruction (LLM)

> On the second slide, the image looks a bit too bright—could you slightly reduce its brightness so it's darker, but keep everything else the same (no cropping, resizing, or other edits)? When you're done, please export just that image as a PNG and save it on the Desktop as `/home/user/Desktop/background.png`. Thanks!

Issue: The clarified instruction inverts the original task due to the confusing evaluation function name; the evaluation JSON checks that the original photo is dimmer than the new photo rather than checking that the new photo is brighter than the original.

Human-Corrected Instruction:

> On the second slide, the image looks a bit too dim—could you slightly enhance its brightness so it's brighter, but keep everything else the same (no cropping, resizing, or other edits)? When you're done, please export just that image as a PNG and save it on the Desktop as `/home/user/Desktop/background.png`. Thanks!

Example 4: Human-corrected trivial task after clarification

Original Instruction: *Find discussions of community and open one with most replies.*

Clarification Context

- Evaluation JSON with arguments to evaluator function `is_expected_active_tab`
- Implementation of `is_expected_active_tab` function
- Initial screenshot (google.com)

Clarified Instruction (LLM)

> Go to FlightAware's Community/Discussions forum and open the topic with the most replies—The Banter Thread—so that the active tab is https://discussions.flightaware.com/t/the-banter-thread/4412.

Issue: The clarified instruction includes the ground-truth URL that is checked which trivializes the search process in the task.

Human-Corrected Instruction:

> Go to FlightAware's Community/Discussions forum and open the topic with the most replies on the entire website.

D.2 Retry (Clarify) with User Simulator

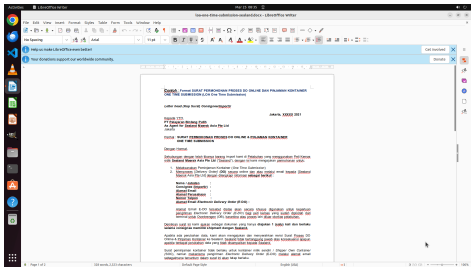
We present three qualitative examples (one per model) where the user simulator's feedback resolved a genuine ambiguity in the task instruction, enabling the agent to recover from failure. In each case, the agent's first interpretation was reasonable but did not match the evaluator's expectation; the user simulator clarified the intended meaning without providing step-by-step instructions.

Example 1: Ambiguous scope:document-level vs. global default (GPT-5)

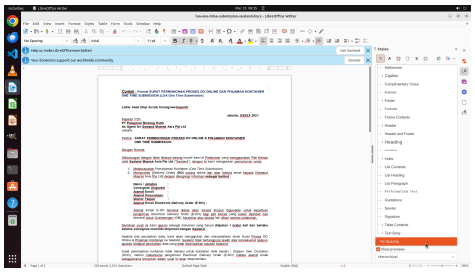
Instruction: *"Make Times New Roman the default Font."*

Outcome: Pass after 1 retry (32 total steps).

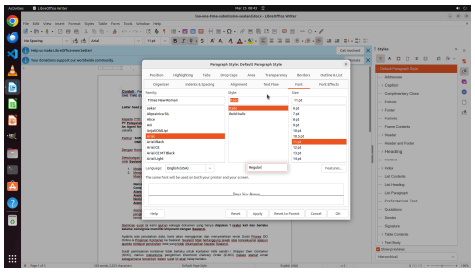
First attempt (failed): The instruction does not specify whether "default" means the default for the current document or the global LibreOffice Writer default for all future documents. The agent reasonably changed the Default Paragraph Style in the open document: a valid interpretation that only affects that file.



(a) Initial state



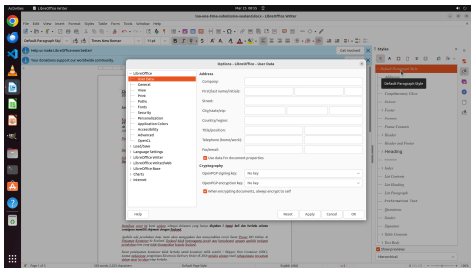
(b) Open Styles sidebar (F11)



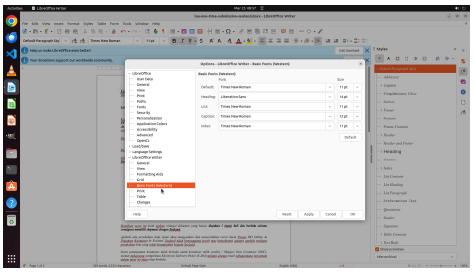
(c) Set font in Paragraph Style dialog

> **User Simulator Feedback**
>
> You changed the "Default Paragraph Style" in the open document, which only affects that file. The global Writer default wasn't updated: in `~/.config/libreoffice/.../registrymodifications.xcu`, the default font is not set to "Times New Roman". Update LibreOffice Writer's global "Default" font (Basic Fonts/Western) so that this key's value becomes "Times New Roman".

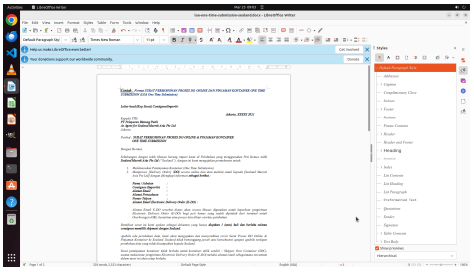
After feedback (succeeded): The user simulator clarified that "default font" means the *global* setting, not the document-level style. The agent navigated to Tools → Options → Basic Fonts (Western) and updated the system-wide default.



(d) Open Tools → Options



(e) Set Basic Fonts (Western)

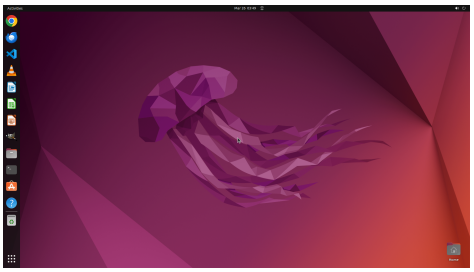


(f) Confirm and call `done()`

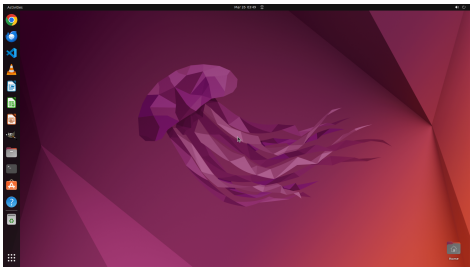
Example 2: Ambiguous setting name:`idle-dim` vs. `idle-delay` (Claude Sonnet 4.6)

Instruction: *"Could you set the 'Dim screen when inactive' to off in setting?"*
Outcome: Pass after 1 retry (16 total steps).

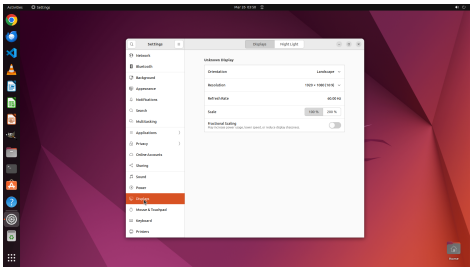
First attempt (failed): The instruction references "Dim screen when inactive," which maps naturally to the GNOME setting `idle-dim`. The agent opened Settings, then used the terminal to run `gsettings set org.gnome.settings-daemon.plugins.power idle-dim false`: a reasonable interpretation. However, the evaluator checks `org.gnome.desktop.session idle-delay`, which controls the inactivity timeout rather than the dimming behavior.



(a) Initial: Ubuntu desktop



(b) Screenshot (no visible change)

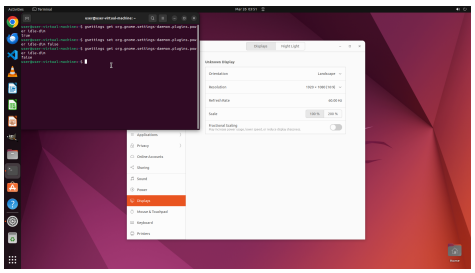


(c) Settings open on Display page

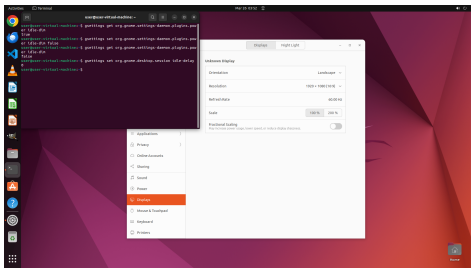
User Simulator Feedback

You changed the wrong setting. You set `org.gnome.settings-daemon.plugins.power idle-dim` to `false`, but the required key is `org.gnome.desktop.session idle-delay`, which must be `0` (expected output: "uint32 0"). Because `idle-delay` wasn't set to 0, the check fails. Update `idle-delay` to 0.

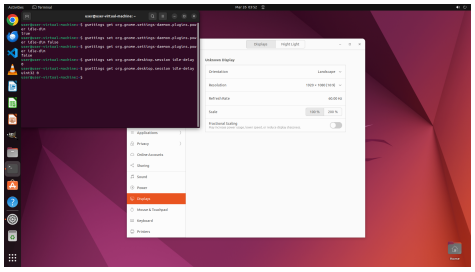
After feedback (succeeded): The user simulator specified the exact gsettings key that needs to be changed. The agent ran `gsettings set org.gnome.desktop.session idle-delay 0` in the terminal.



(d) Terminal: ran `idle-dim false`



(e) Runs `gsettings set idle-delay 0`

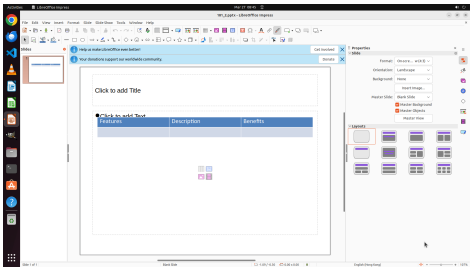


(f) Verifies `uint32 0`, calls `done()`

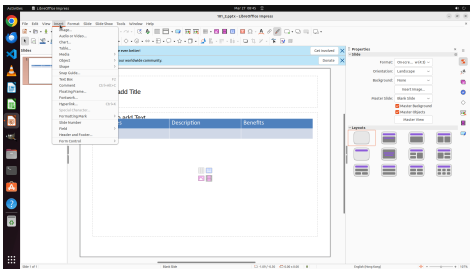
Example 3: Ambiguous term:"note" means speaker notes, not comment (Kimi K2.5)

Instruction: *"Add a note 'APP' into the slide and give the slide a purple background color."*
Outcome: Pass after 1 retry (16 total steps).

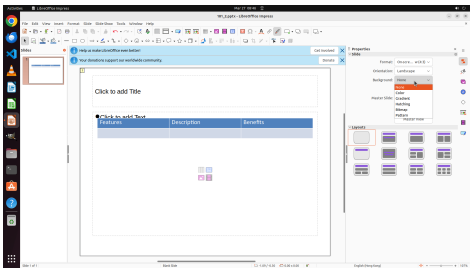
First attempt (failed): The word "note" in LibreOffice Impress is ambiguous: it could refer to a *comment* (an annotation visible on the slide), a *text box*, or *speaker notes* (the notes pane below the slide used during presentations). The agent added a comment annotation: a reasonable interpretation. The evaluator expected speaker notes.



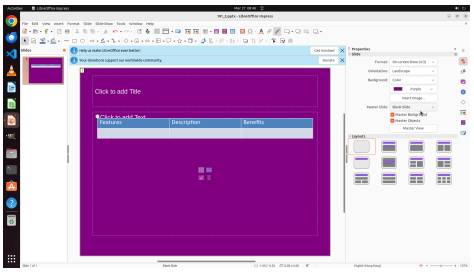
(a) Initial: slide in Impress



(b) Open Insert menu (Comment option)



(c) Set background to purple

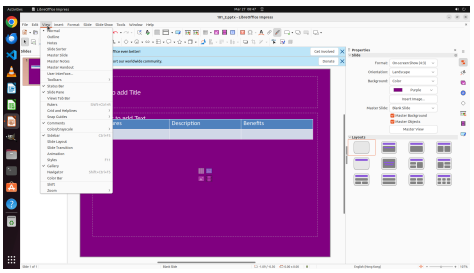


(d) Purple background applied, calls done()

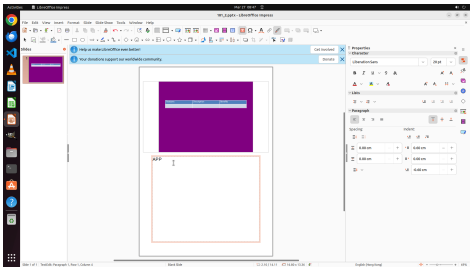
User Simulator Feedback

You added a comment on the slide instead of a slide note. The expected change is that the slide's Notes (speaker notes) contain exactly "APP" (case-sensitive). The purple background color looks correct. Click into the Notes pane below the slide and type "APP" there.

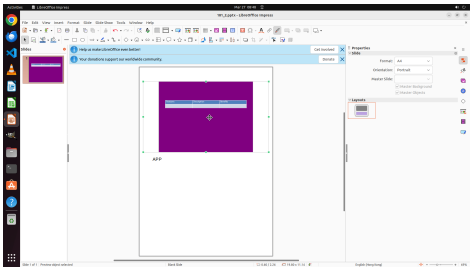
After feedback (succeeded): The user simulator disambiguated "note" as speaker notes. The agent clicked into the Notes pane at the bottom of the Impress window and typed "APP."



(d) Open View menu → Notes



(e) Type "APP" in Notes pane



(f) Notes pane shows "APP", calls done()

E Clarification Details

To study the effect of reducing instruction ambiguity, we construct clarified versions of task instructions by making minimal edits that align the instruction with the evaluator's success criteria. Given the original instruction, the current environment state (screenshot), the task configuration (evaluation JSON), and the associated evaluator implementation, we prompt an LLM (GPT-5) to identify aspects of the evaluator that are not explicitly specified in the instruction and to produce a minimally clarified version. The prompt enforces that clarifications preserve the original intent and nat-

ural, human-like tone, while only adding details required by the evaluator (e.g., exact file names, formats, or locations). The full prompt is provided in Appendix C.2.1.

To ensure that clarification improves alignment with evaluation criteria without introducing unintended artifacts, we perform a verification and human correction procedure. For each task, we evaluate both the original and clarified instructions using multiple independent runs (three runs each with GPT-5 and Claude Sonnet 4.6) and compare the resulting success rates. We then use a modified version of Behavior Judge (Gonzalez-Pumariega et al., 2026) to compare trajectories from both settings (Appendix C.2.2), conditioned on the evaluator implementation, and identify whether changes in success rate are due to improved alignment, random variation, or issues introduced by clarification (e.g., making a task impossible or overly trivial) (Appendix C.2.3). We flag cases with substantial performance shifts (e.g., consistently solved to never solved and vice versa). These cases are then manually reviewed, using the judge's analysis as guidance, the raw trajectories, and the evaluator requirements to guide minimal edits that correct problematic instructions. In total, we manually corrected 25 out of 361 tasks, including 20 cases where clarification introduced impossible constraints and 5 where it made the task overly trivial. This procedure ensures that the clarified instructions better reflect evaluator requirements while preserving task difficulty. Examples of corrections are shown in Appendix D.1.

F Environment Perturbation Details

Property	Default	Set 1	Set 2
Wallpaper	Jammy Jellyfish	Jammy Jellyfish	Optical Fibers
Cursor size	24 px	48 px	16 px
Dock position	Left	Bottom	Right
Icon theme	Yaru	Adwaita	Humanity
Timezone	America/Los_Angeles	Asia/Tokyo	Europe/London

Table 5: Perturbation sets. Each row describes one modified desktop property. All modifications are purely cosmetic and do not affect application functionality or task correctness.