VERICUA: VERIFICATION AS AGENTIC EXPLORATION FOR COMPUTER-USING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-using agents (CUAs) follow natural-language instructions on desktops, mobile devices, and web browsers by reading and acting within the OS. Their evaluation, data curation, and reinforcement learning all require verifying at volume whether each trajectory fulfilled its instruction. Hand-written checkers stop at benchmark scale and human annotation cannot keep pace, so the field increasingly turns to VLM judges and automatically generated verifiers, each with its own reliability problem, from the false positives of judges to checks never grounded in the environment. Both rest on a static verifier that sees only partial information, while the evidence of success may be scattered far beyond that view. To overcome this bottleneck, we present VeriCUA, a new paradigm for verifying CUA tasks in which the verifier adapts to and actively explores its environment for evidence of task completion. We equip it with an adaptive, dynamic action space combining code and GUI to navigate recorded trajectories and online environments alike. VeriCUA decomposes the task into a rubric of end-state requirements, plans and explores at test time to gather evidence for each, and types each piece by how it was obtained. Only attested effects, never the agent's own assertions, satisfy a requirement, and a deterministic arbiter then derives the verdict from this evidence. Experiments show VeriCUA to be a promising solution across online and offline settings. On recorded trajectories across multiple testbeds and benchmarks, it surpasses typical judging methods and improves markedly on hard cases. On online environments, it certifies far fewer failed runs as successes than the agent's own completion claim and agrees more often with official evaluators. Analyses trace the gain to the action space, find that it grows with a stronger backbone, and show where recorded evidence runs out. Its evidence-backed verdicts are precise enough for scalable generation of CUA reward signals.

1 INTRODUCTION

Computer-using agents (CUAs) powered by vision-language models (VLMs) operate desktops, mobile devices, and web browsers (Xie et al., 2024; Zhou et al., 2024; Rawles et al., 2025). Given a natural-language instruction, an agent reads the screen, plans, and acts until it deems the task complete. The run leaves a trajectory, the interleaved record of its thoughts, actions, and observed states. Improving these agents depends on one judgment made at volume: whether that trajectory actually fulfilled its instruction. Evaluation, data curation (Sun et al., 2025), and reinforcement learning (Xue et al., 2026b) all rely on this judgment.

Obtaining this judgment at scale, however, remains an open problem for recorded trajectories and online environments alike. Success requires every condition the instruction places on the final state to hold, and some of these conditions never appear on screen. Benchmark suites therefore grade success with hand-written checkers, which do not scale and run only on the online environment after the task ends (Xie et al., 2024; Rawles et al., 2025; Xie et al., 2025). Trajectories collected from real-world websites and applications have no checker at all. Human annotation closes the quality gap but cannot keep pace with training-scale volume (Sun et al., 2026a), so reliable reward signals remain scarce. The field has turned to automated verifiers that scale. The VLM judge (Pan et al., 2024; Xue et al., 2026b) reads the trajectory in a single model call and returns a verdict, and automatically generated verifiers (Xue et al., 2026a) let a model write the checker. Other verifiers read one platform's native

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

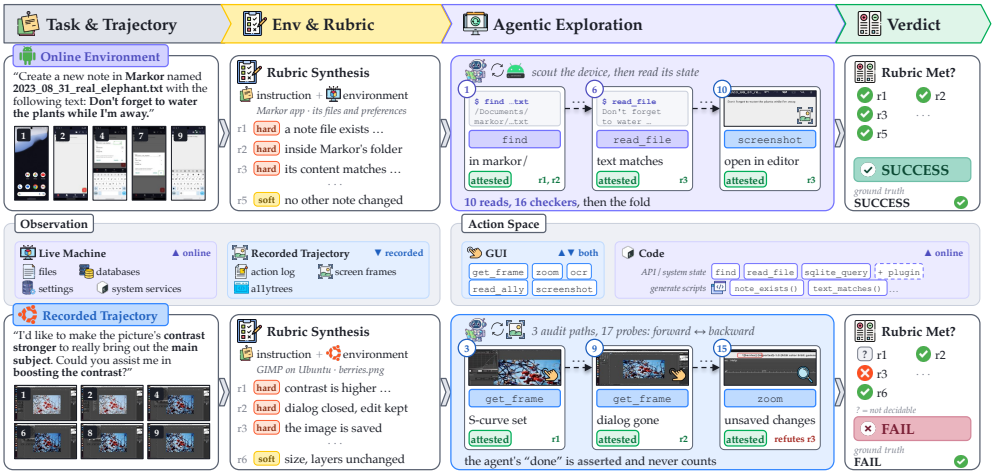

Figure 1: VeriCUA casts verification as agentic exploration. A rubric of requirements guides the actions that gather evidence, online or from a recording, and attested findings alone decide the verdict.

signal, such as the web DOM (Chae et al., 2025) or the system state of a mobile device (Sun et al., 2026b).

Whether these automated verifiers deserve the trust is a different question. Measured against human-labeled trajectories, judges share a leniency bias and fall short on the hard cases (Sun et al., 2026a). Their signature error is the *false positive*, a failed run certified as complete because the agent presented it as complete. The cause, we argue, lies in what the verifier gets to see. A trajectory narrates what the agent intended, whereas success is a property of the environment's final state, and over a long horizon the two might diverge. An automatically generated verifier does read the final state, but through a single script written without first mapping where that state lives. Both are static verifiers that see only partial information, and more test-time compute (Snell et al., 2024; Brown et al., 2024) only re-reads the same partial view.

To overcome this bottleneck, we present VeriCUA, a new paradigm for verifying CUA tasks. Whether a task has been completed is hard to judge as a whole, so the verifier acts as an agent that explores its environment and makes full use of the information it holds (Figure 1). It first synthesizes an environment-aware rubric of fine-grained hard and soft requirements on the final state. One open-ended judgment thus becomes a series of subproblems, each small enough to settle with evidence. The verifier then solves each subproblem by test-time exploration and interaction over a hybrid action space, choosing GUI or code actions by what earlier reads returned and reaching evidence beyond what the screen shows. Because every action can bring new evidence into view, verification gains a test-time scaling axis that a single-pass judge lacks. Finally, a strict evidence discipline lets only an *attested* effect, and never the agent's own account, satisfy a requirement. A deterministic arbiter derives the verdict from this evidence, and it may abstain rather than force a false positive or negative. The exploration is agentic; the adjudication deliberately is not.

We take a unified view of rewarding CUAs across settings and platforms. Whether on the web, a mobile device, or a desktop, a recorded trajectory or an online environment may hold evidence of the task's final state, and the verifier can go looking for it. To realize this view, we design an action space that grows with state access. An online environment lets the verifier reach the live machine itself, including persisted state that never appears on screen. The verifier writes its verification scripts there as allowlisted read-only checkers, and their findings pass through the same evidence discipline. Recorded trajectories are static and offer more limited observations, yet the verifier can still explore them with GUI actions such as frame lookup, zooming, and OCR. The actions available on a recording form a subset of those available online, so test-time interaction reaches more evidence online. One verification process can thus filter recorded trajectories for training data and supply reward signals for reinforcement learning online.

Experiments bear this out in both settings. In online environments, VeriCUA agrees with gold more closely than any baseline, certifies fewer failed runs as successes, and is right on about nine in ten of the episodes it decides. Combining further verifier passes raises its decision rate and decided accuracy together. On recorded trajectories across multiple testbeds and benchmarks, VeriCUA surpasses typical judging methods (Sun et al., 2025; Xue et al., 2026b; Li et al., 2026), and the gains concentrate on the hard cases where judges fail most. The gain grows further with a stronger backbone. An identifiable subset of its failure verdicts is right about 90% of the time across three gold standards, precise enough to filter training data, and the same verdicts surface errors in the benchmarks' own gold labels. Together, these results point to VeriCUA as a promising universal solution across online and offline verification that scales like automatically generated verifiers without giving up reliability.

Our primary contributions are as follows:

- **A New Paradigm for Verifying CUA Trajectories.** We present VeriCUA, which recasts verification as agentic exploration for evidence that a task's rubric is met, under one unified view of recorded trajectories and online environments across platforms.
- **Adaptive Hybrid Evidence Gathering.** An adaptive, dynamic action space combining code and GUI makes each rubric requirement solvable through test-time exploration and interaction, and a deterministic arbiter derives reliable verdicts from harness-typed evidence.
- **Experiments and Analyses.** Across testbeds and benchmarks VeriCUA surpasses typical judging methods, with the largest gains on hard cases, and in online environments it certifies fewer false positives than existing judges and verifiers. Analyses reveal where the gains come from.

2 Related Work

Computer-Using Agents and Their Evaluation. Computer-using agents powered by vision-language models now operate web, mobile, and desktop software from natural-language instructions (Zhou et al., 2024; Rawles et al., 2025; Xie et al., 2024), and dedicated environments extend them to professional and scientific workflows (Sun et al., 2026c). Their progress rests on trajectories. Synthesis pipelines construct them at scale and filter them before training (Sun et al., 2025), continual reinforcement learning turns them into reward (Xue et al., 2026b), and safety studies audit them for system-level violations (Sun et al., 2026b). Every one of these uses consumes a judgment about whether a trajectory succeeded, and benchmarks supply it with hand-written programmatic checkers, which cover only the outcomes their authors anticipated and whose gold labels keep failing audits (Xie et al., 2025). Trajectories collected beyond a benchmark have no checker at all.

Verifying Trajectories and Reward Models. The scalable alternatives to checkers and annotators are learned or prompted verifiers. VLM judges read a trajectory in a single call and return a verdict (Pan et al., 2024; Xue et al., 2026b), trajectory reward models score it for filtering (Sun et al., 2025), and trained critics distill the judging role into open reward models (Li et al., 2026). Reward benchmarks then measure what the reported agreement conceals: against cross-platform human gold, judges share a leniency bias and certify failed runs as successes (Sun et al., 2026a; Lin et al., 2025). Automatically generated verifiers write the checker with a model instead of by hand (Xue et al., 2026a). Judges that act have appeared as well, re-inspecting recorded screenshots through an iterative controller (Wan et al., 2026) or probing the online environment with write-capable tools (Cui et al., 2026). These designs share the same limitations. The verdict rests on whatever surface the judge sees, while the deciding evidence often sits in files or stores. The judge is briefed with the agent's own account, which is the channel under test. Uncertainty surfaces only as self-reported confidence, and a verifier allowed to write may alter the state it checks. VeriCUA inverts each of these with a rubric grounded in the task and its environment, read-only exploration across surfaces and settings, and a verdict resting on attested evidence alone.

3 Method

3.1 Verification as Active Evidence Gathering

An executing agent follows an instruction g in an environment, producing a trajectory $\tau = [o_0, a_0, \ldots, o_T]$ of observations, actions, and thoughts and leaving the environment in a final state

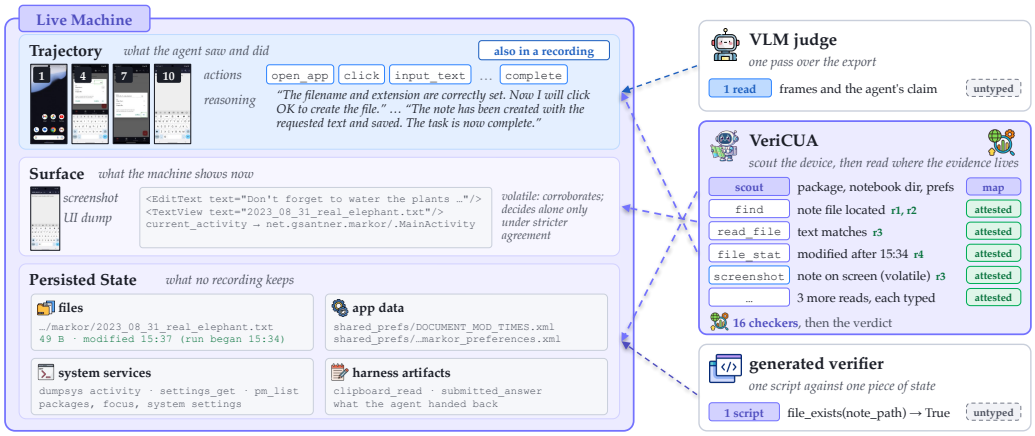

Figure 2: VeriCUA reads task state at all three layers, from the trajectory a recording exports to the surface and the persisted state that only the live machine holds.

s_T. Verification asks whether s_T satisfies g and returns a verdict $y \in \{\text{SUCCESS}, \text{FAIL}\}$. The verifier never sees s_T directly. It sees a set of state sources $\mathcal{S} = \{S_1, \ldots, S_m\}$, each a partial and imperfect view of s_T. A source may be the actions, reasoning, or screen frames of a recorded trajectory, the accessibility trees (a11ytrees) when the platform kept them, a frozen snapshot of the final environment, or the live machine itself in an online environment.

VeriCUA treats verification as agentic exploration and models it as a sequential decision process over these sources. The verifier is an agent with a read-only action space $\mathcal{A}(\mathcal{S})$ determined by the sources a setting exposes. An action $\alpha = (v, x)$ pairs a verb with its arguments, and the harness executes it as a function of the exposed sources, $\alpha(\mathcal{S}) \to (o, t)$, returning an observation o and an evidence type t. Because every action reads and none writes, s_T does not change while it is examined, and each action only reduces the verifier's uncertainty about it. Given the rubric R of Section 3.2, a policy π_V chooses the next action from the instruction, R, and the typed evidence $\mathcal{E}_k$ gathered so far, until every hard requirement is settled or the evidence budget is spent. This loop is the verifier's test-time exploration. A deterministic arbiter, a fold $F(\mathcal{E}) \to y$ over this evidence, then derives the verdict. Because the policy interacts with a source only through the pairs (o, t) it returns, the same procedure runs on screenshots, on a11ytrees, or on both, and in either setting. What differs is the action space $\mathcal{A}(\mathcal{S})$, which grows with state access, so the actions available on a recorded trajectory form a subset of those available online (Section 3.3).

3.2 Environment-Aware Rubric Synthesis

The first stage turns the instruction into a rubric grounded in the environment. The verifier maps the instruction g and the environment's static context c to a rubric $R = D(g, c) = \{(r_i, h_i)\}_{i=1}^{n}$, where each requirement r_i is an atomic predicate on the final state s_T and $h_i \in \{\text{hard}, \text{soft}\}$ marks whether it gates the verdict or refines it. Each r_i states an end state that the instruction demands, so what the agent claims about its run cannot redefine success. The verifier then explores the environment, builds a map of where the state named by each requirement lives, and grounds every requirement in that map. The grounded rubric turns one open-ended judgment into subproblems, each small enough to settle with evidence (Figure 1). Because these subproblems are independent, much of the exploration runs in parallel, which keeps the wall-clock cost of test-time exploration low. Each requirement ends the audit as met, unmet, or unknown, and the arbiter of Section 3.4 derives the verdict from them.

3.3 Adaptive Hybrid Action Space

A recorded trajectory keeps the agent's observed states, as screen frames and on some platforms a11ytrees, together with its actions and reasoning. An online environment adds the live machine behind it, whose current surface and persisted layer no recording keeps. That layer is where decisive

evidence tends to live, and today's checkers, hand-written or generated, read one piece of it per task. Figure 2 shows the three layers, and Table 5 in Appendix A lists every verb.

The action space is hybrid in that GUI reads of frames and trees sit beside code-level reads of files, databases, content providers, and system services. In an online environment the verifier can also write a verification script, a getter plan with one predicate that the harness executes as a single action. The verifier can thus reach evidence wherever it lives and pick for each requirement the source able to settle it. The space is adaptive in that the verifier chooses each read from what earlier reads returned, and a targeted probe re-reads only the hard requirements that remain unsettled. Since each action costs one call and yields one typed observation, what the verifier can settle depends on its action space more than on its backbone. The harness computes the type t from the verb, its arguments, and the source metadata, such as which frame or store was read and whether it came before or after the decisive action, so the model never chooses it.

Non-Interference. Nothing stops a verifier from acting on the machine once the episode ends or the agent's step budget runs out; it could open the document under check or re-run the query behind an answer. Every write risks turning the state under test into one the verifier itself produced, so VeriCUA keeps its verbs to reads and enforces that mechanically in both settings, as described in Appendix A.

3.4 Evidence Discipline

Three rules turn the gathered evidence into a verdict. *Grounded.* A finding must cite the action that produced it; a citation matching no executed call is rejected as fabricated, and the type comes from what that call read. *Asymmetric.* Types partition into an attested class, which witnesses the persisted effect, and an asserted class, which expresses intent, such as the agent's own claim or a frame taken before the decisive action (Table 5). Attested evidence is necessary before any requirement counts as satisfied, and asserted evidence never suffices. Each setting adds a label of its own, frame timing on a recording and persistence on a machine, which limits what a single observation may decide. *Deterministic.* The fold F marks a requirement met only on attested evidence and unmet only with corroboration across independent probes. Anything still unsettled is resolved by an initial judgment that the verifier made before the audit and kept sealed from it, or the verifier abstains rather than force a verdict the evidence cannot support. An abstention withholds the signal instead of supplying a wrong one, so no forced verdict reaches a training or filtering pipeline as reward. Because F is a pure function over stored findings, its thresholds can be re-tuned offline.

3.5 From Online Environments to Recorded Trajectories

Both settings follow the same procedure of rubric synthesis, test-time exploration, and adjudication, and they differ only in how exploration spends the actions each setting exposes. In the online setting, the map records the packages, stores, and schemas of the live machine, and VeriCUA spends its actions on verification scripts, allowlisted read-only checkers written over that map. A requirement is met by one persistent attested read, and an unsettled one is folded to FAIL or reported as an abstention. Because checkers attach to the task, one synthesis serves every rollout of it, so in this setting VeriCUA scales like an automatically generated verifier while the evidence discipline keeps its verdicts reliable. On a recorded trajectory, the map records the steps and frames that concern each requirement, and K independent audit paths gather support forward and hunt refutations backward over those frames. Every path must attest a requirement before it counts as met, and what remains unsettled is resolved by the sealed initial judgment. Appendix A aligns the two settings stage by stage (Table 6).

4 Experiments

4.1 Experimental Setup

Benchmarks. In online environments, we run AndroidWorld (Rawles et al., 2025) on an Android emulator, OSWorld (Xie et al., 2024) in a virtual machine, and ScienceBoard (Sun et al., 2026c) in its scientific desktop environment. Each episode is executed by the benchmark's own agent scaffold, and the verifier probes the resulting machine state after the episode ends. On recorded trajectories,

Table 1: Performance in online environments.

Environment	Judge	Prec	Rec	F1	Acc	False Positive Rate ↓	Decided	Decided Acc
AndroidWorld	VLM judge	79.3	69.7	74.2	78.4	15.4	100%	78.4%
	Auto-Generated verifier	77.8	84.8	81.2	82.4	16.9	100%	82.4%
	VeriCUA	**90.6**	**87.9**	**89.2**	**90.5**	**7.7**	82%	**95.1%**
OSWorld	VLM judge	68.2	75.0	71.4	73.4	27.9	100%	73.4%
	Auto-Generated verifier	70.6	**78.1**	74.2	75.9	25.9	100%	75.9%
	VeriCUA	**80.3**	71.3	**75.5**	**79.5**	**13.9**	78%	**88.7%**
ScienceBoard	VLM judge	42.9	60.0	50.0	71.6	24.8	100%	71.6%
	Auto-Generated verifier	50.9	**72.5**	59.8	76.9	21.7	100%	76.9%
	VeriCUA	**77.4**	60.0	**67.6**	**86.4**	**5.4**	70%	**91.6%**

we evaluate on WebArena (Zhou et al., 2024) rollouts from three executing agents and on OSReward (Sun et al., 2026a), whose human-labeled trajectories span four platforms and several agents.

Protocol. All verifiers and baselines run on `gemini-3-flash-preview`, and the backbone study re-runs them on `gemini-3.1-pro-preview`. In online environments, gold is each benchmark's official evaluator and the executing agents also run on `gemini-3-flash-preview`. On WebArena we provide a11ytree variants of the input beyond the GUI, to the verifier and the judges alike. The remaining protocol details are provided in Appendix B.

Baselines. We compare against a VLM judge, an automatically generated verifier, and two published judges with more structure, with implementation details provided in Appendix B.

- **VLM judge.** The most basic baseline, a classic single-pass judge such as TRM (Sun et al., 2025), which reads the trajectory in one model call and returns a verdict or a score.
- **Auto-Generated Verifier** (Xue et al., 2026a). One machine-written evaluator script per task, run once against the machine for a binary verdict.
- **CUAJudge** (Xue et al., 2026b). A checklist judge that derives key points from the task and checks them against sampled frames in one call.
- **OS-Themis** (Li et al., 2026). A milestone judge that checks ordered milestones and reviews them.

The VLM judge runs in both settings, the auto-generated verifier only in online environments, and CUAJudge and OS-Themis only on recorded trajectories.

Metrics. We report agreement with gold, precision, recall, and F1 for the SUCCESS class, and the false positive rate, with abstentions counted as FAIL. For verifiers that abstain we also report the decision rate and the accuracy on decided episodes. Exact definitions are provided in Appendix B.

4.2 Online Verification

Accuracy and False Positives. Across the three online environments, VeriCUA reaches the highest accuracy and the fewest false positives, and the auto-generated verifier is the stronger of the two baselines (Table 1). The gap is widest on ScienceBoard, where the state of each scientific application is reached only through its own interface and a script written once often reads the wrong place. There the auto-generated verifier certifies four times as many failed episodes as VeriCUA, whose exploration locates the state before reading it. On OSWorld the auto-generated verifier recovers more of the successes, and VeriCUA trails it on recall alone.

Deciding and Abstaining. The two baselines answer on every episode, so their accuracy is also their decided accuracy. VeriCUA instead abstains when the evidence it gathered cannot settle a hard requirement, which happens on between a fifth and a third of the episodes depending on the environment, and on the episodes it does decide it is right about nine in ten times. An abstention records that the checkers reached the state but could not settle a requirement from what they read, or

that independent reads disagreed. Folded to FAIL, abstentions cost recall but never precision, which is the trade the reward model use case wants.

4.3 Offline Verification

Table 2: Performance on recorded trajectories.

Judge	OSReward-Hard		OSReward		WebArena		AndroidWorld		OSWorld	
	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc
TRM	57.4	64.8	82.2	83.1	79.6	79.8	88.3	89.6	80.3	73.7
CUAJudge	63.4	62.5	83.7	82.6	81.3	81.8	91.2	90.7	83.4	78.2
OS-Themis	65.7	63.2	80.2	79.8	75.4	74.3	89.4	89.0	82.1	79.0
VeriCUA	**70.4**	**68.3**	**87.2**	**86.3**	**82.7**	**83.1**	**94.3**	**94.1**	**86.3**	**81.2**

Across Benchmarks and Difficulty. On recorded trajectories, VeriCUA leads every judge on both metrics in every column of Table 2, on human labels and on official checkers alike. On OSReward the advantage grows with difficulty. From the full set to the hard subset, the two strongest single-call judges lose more accuracy than VeriCUA does, so the evidence discipline holds up where judging is hardest. These are the trajectories on which a single reading of the recording settles the least, and where exploration has the most to add.

Robustness and the Recorded Ceiling. The lead does not depend on who produced the trajectories or on what the judges read. On WebArena it holds for each of the three rollout agents, as the per-agent breakdown in Table 15 of Appendix C shows, when the judges receive a11ytrees alone, and when they run on the stronger backbone (Section 5), and on recorded AndroidWorld and OSWorld trajectories VeriCUA leads every judge on both metrics as well. Recorded evidence has a ceiling, though. With no machine to probe, VeriCUA falls back to its initial judgment on the trajectories whose requirements no frame captures, 23% of them on WebArena, and Appendix C maps where that ceiling sits. For investigating stability, we provide experiemnts in appendix D, which walks through five episodes, two online and three recorded, with the evidence each verdict rests on.

5 Analysis

5.1 GUI Versus Hybrid Action Space

GUI Only versus Hybrid. Restricting VeriCUA to GUI actions on online AndroidWorld leaves its false positives unchanged but nearly halves the share of episodes it decides, and recall falls with it (Figure 3). The requirements that block a decision are those whose evidence sits in files and stores rather than on screen, so the code channel buys coverage at no cost in precision. In the paired study, every win unique to the code channel decided on a store or file read, and none on the screen.

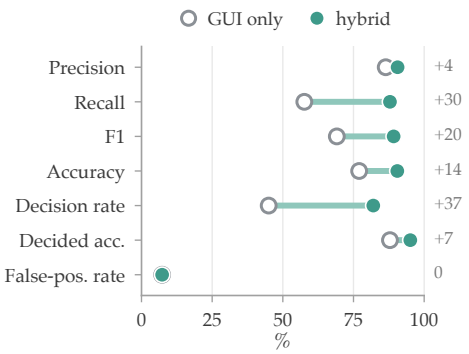

Figure 3: Each metric of VeriCUA with GUI actions alone and with the hybrid action space on the online AndroidWorld suite. Hollow markers are GUI only, filled markers the hybrid space, and the column at the right gives the change.

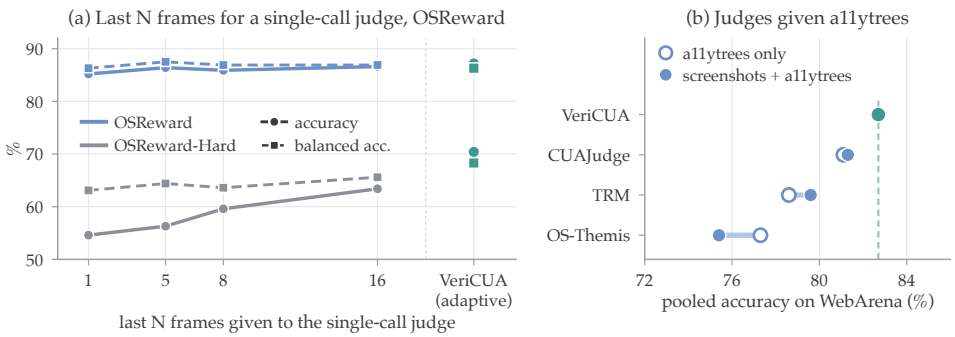

Figure 4: More observation for a single-call judge. (a) The OSReward judge reading the last N frames of each trajectory, with VeriCUA at the right; OSReward's default is five. (b) The WebArena judges reading a11ytrees alone or with screenshots, pooled over three agents, against VeriCUA (dashed).

5.2 The Effect Of Observation

Different Observations. On WebArena, the observation the verifier reads changes how much it decides on its own. With screenshots alone it commits to a third of the trajectories, and adding a11ytrees raises the share to three quarters at a small increase in cost while accuracy stays level. Given a11ytrees alone, it decides the most at the lowest cost but gives up a few points of accuracy. Table 14 in Appendix C gives the full breakdown. The share of verdicts that rest on structured evidence rises from none to nearly all across the three inputs, so structured state buys coverage, and the verdicts it enables are as reliable as those from frames alone. The judges do not gain the same way. Given a11ytrees alone or together with screenshots, each moves by about a point in either direction, and VeriCUA stays ahead of every judge under both inputs (Figure 4b). The per-agent values are provided in Table 16 of Appendix C.

More Observations. Giving a single-call judge more of the recording is the obvious alternative to letting a verifier explore it, so we hand the OSReward judge the last one to sixteen frames of every trajectory (Figure 4a). On the full set the extra frames move it by a point or two around VeriCUA, and on the hard subset they lift its accuracy by about nine points while leaving it seven points behind. The frames a judge can be given are still the frames the recording happens to hold; exploration reads the ones that settle each requirement and stops when they do. Scaling the observation naively therefore does not substitute for agentic exploration.

5.3 Test-Time Exploration

Combining Passes. Verification also scales with the passes the verifier spends. On the multi-pass AndroidWorld suite, three independent draws of one configuration and one draw with a vision fallback share one restore per episode. The combination rule takes the decided verdict and abstains when two

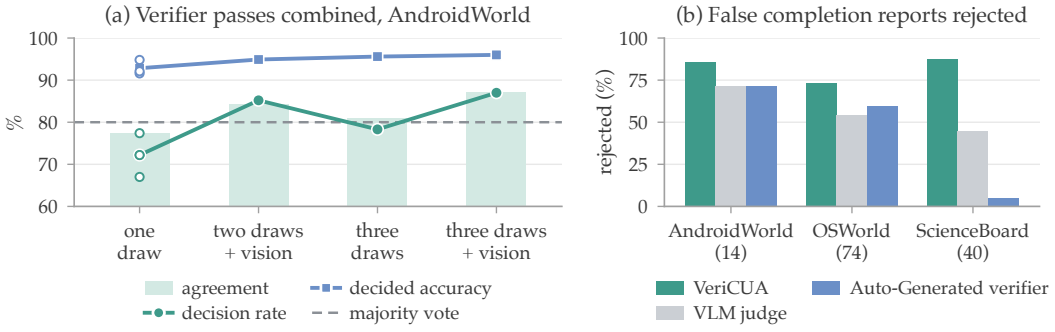

Figure 5: (a) Combining verifier passes raises agreement, decision rate, and decided accuracy together. (b) The share of the agent's false completion reports that each verifier rejects.

decided verdicts disagree. Adding identical draws buys little, whereas adding the pass that reads a different channel buys most of the gain. Three draws plus the vision pass lift agreement and the decision rate to 87% while decided accuracy also rises (Figure 5a). The rule also keeps wrong-FAIL verdicts an order of magnitude below majority voting over the same draws, as Table 10 in Appendix C reports.

Matched Budget. Exploration spends more calls than a single pass, so we also compare the methods at an equal budget of steps and screens seen, all on the flash backbone (Table 3). TRM is run as a majority vote over ten draws, CUAJudge is given the same number of steps, and VeriCUA's exploration is capped at the same budget. VeriCUA stays ahead of both judges on OSReward and on OSReward-Hard, a point or two below its accuracy with the default budget, so exploration turns a fixed budget into more accuracy than one pass or repeated passes do.

Table 3: Accuracy at a matched budget of steps and screens seen, flash backbone.

Judge	OSReward	OSReward-Hard
TRM (N=10)	83.5	58.5
CUAJudge	84.8	64.5
VeriCUA	**86.4**	**68.7**

Catching the Agent's Own Report. The agent's own completion report is the weakest signal of all. Across the three online environments the agent reports completion on 342 episodes (AndroidWorld on a subset), and 128 of those reports contradict gold (Figure 5b). VeriCUA rejects about four in five of the false reports, the VLM judge about half, and the auto-generated verifier varies from most of them on OSWorld to almost none on ScienceBoard, where its scripts read the state the agent's report describes rather than the state the application holds. Figure 7 in Appendix D walks through one such report, refuted from the media store.

5.4 Gains From a Stronger Backbone

Moving every method from the flash to the pro tier separates the frameworks by how much of the stronger model they can use (Table 4). On OSReward-Hard VeriCUA gains close to three points of accuracy and also improves in balanced accuracy, whereas the two judges barely move or lose ground. A single pass that rereads the same frames has nowhere to spend the extra capability. In the online AndroidWorld environment every framework gains and VeriCUA stays ahead on both tiers, as Table 12 in Appendix C shows.

Table 4: Accuracy and balanced accuracy on OSReward-Hard with each framework on the flash and pro tiers of the same model family.

Judge	Acc			BalAcc		
	Flash	Pro	Δ	Flash	Pro	Δ
CUAJudge	63.4	62.7	−0.7	62.5	58.4	−4.1
OS-Themis	65.7	66.1	+0.4	63.2	61.1	−2.1
VeriCUA	**70.4**	**73.1**	**+2.7**	**68.3**	**69.2**	**+0.9**

Beyond these analyses, Appendix C explores three further questions. It reports which of the arbiter's evidence rules carry the result and how recall falls with the number of hard requirements, measures the precision of VeriCUA's confident failure verdicts and audits the gold-label errors they surface, and maps where recorded evidence runs out on OSReward-Hard.

6 Conclusion

We presented VeriCUA, a new paradigm that verifies computer-using agents by agentic exploration. For each task the verifier synthesizes a rubric of hard and soft requirements grounded in the environment. It settles each requirement by acting through a hybrid action space of GUI and code reads and derives the verdict from attested evidence alone. One procedure serves recorded trajectories and online environments on web, mobile, and desktop platforms, and the recorded action space is a subset of the online one. In online environments VeriCUA certifies fewer failed episodes as successes than a VLM judge or an automatically generated verifier, and on recorded trajectories it surpasses typical judging methods with the largest gains on the hardest cases. The gain comes from the action space, grows with a stronger backbone and with the state the verifier can reach, and rises further with additional verifier passes. Its evidence-backed verdicts are precise enough to filter training data and to supply reward signals for reinforcement learning.

AI USE STATEMENT

We used generative AI tools to assist with writing and editing the manuscript (grammar checking and language polishing) and to help identify related work. Also, we leveraged AI coding agents to help developing. All technical content, experiments, and claims were produced and verified by the authors.

ETHICS STATEMENT

In online environments, VeriCUA runs against emulators and virtual machines that contain no user data. Every action in its action space is read-only, enforced by an allowlist and a before-and-after digest of the state it touches. All trajectories and labels used in this work come from public benchmarks. We see no direct risk of harm from the method itself. Its main effect is to lower the cost of judging agent behavior, which supports the safe development of computer-using agents.

REPRODUCIBILITY STATEMENT

We will release the source code of VeriCUA for both settings, the baseline implementations, and the evidence records behind every table upon publication. The environments and harness are described in Appendix A. The datasets, baselines, and configurations are given in Section 4.1 and Appendix B. The prompts are in Appendix F.

REFERENCES

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Hyungjoo Chae, Sunghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, Dongwook Choi, Minseok Kang, Gwanhoon Im, ByeongUng Cho, Hyojun Kim, Jun Hee Han, Taeyoon Kwon, Minju Kim, Beong woo Kwak, Dongjin Kang, and Jinyoung Yeo. Web-shepherd: Advancing PRMs for reinforcing web agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=G2kMroO9UV.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. Agentic reward modeling: Verifying gui agent via progressive trajectory-grounded interaction, 2026. URL https://arxiv.org/abs/2602.00575.

Zehao Li, Zhenyu Wu, Yibo Zhao, Bowen Yang, Jingjing Xie, Zhaoyang Liu, Zhoumianze Liu, Kaiming Jin, Jianze Liang, Zonglin Li, Feng Wu, Bowen Zhou, Zun Wang, and Zichen Ding. Os-themis: A scalable critic framework for generalist gui rewards, 2026. URL https://arxiv.org/abs/2603.19191.

Haojia Lin, Xiaoyu Tan, Yulei Qin, Zihan Xu, Yuchen Shi, Zongyi Li, Gang Li, Shaofei Cai, Siqi Cai, Chaoyou Fu, Ke Li, and Xing Sun. Cuarewardbench: A benchmark for evaluating reward models on computer-using agent, 2025. URL https://arxiv.org/abs/2510.18596.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=NPAQ6FKSmK.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025.

Qiushi Sun, Kanzhi Cheng, Yian Wang, Bowen Yang, Hang Yan, Liheng Chen, Fangzhi Xu, Zichen Ding, Nuo Chen, Jialin Cao, et al. Osreward: Instituting standardized evaluation for cross-platform computer-use reward models. *arXiv preprint arXiv:2607.28609*, 2026a.

Qiushi Sun, Mukai Li, Zhoumianze Liu, Zhihui Xie, Fangzhi Xu, Zhangyue Yin, Kanzhi Cheng, Zehao Li, Zichen Ding, Qi Liu, Zhiyong Wu, Zhuosheng Zhang, Ben Kao, and Lingpeng Kong. OS-sentinel: Towards safety-enhanced mobile GUI agents via hybrid validation in realistic workflows. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9529–9553, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-390-6. URL https://aclanthology.org/2026.acl-long.431/.

Qiushi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao, Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, Jianing Wang, Qintong Li, Xiangru Tang, Tianbao Xie, Xiachong Feng, Xiang Li, Ben Kao, Wenhai Wang, Biqing Qi, Lingpeng Kong, and Zhiyong Wu. Scienceboard: Evaluating multimodal autonomous agents in realistic scientific workflows. In *The Fourteenth International Conference on Learning Representations*, 2026c. URL https://openreview.net/forum?id=bJvwJahJeF.

Yang Wan, Zhenhao Zhang, Jierui Wang, and Linchao Zhu. Seekjudge: A practical reward framework for reinforcement learning in computer-use agents. *arXiv preprint arXiv:2607.23263*, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=tN61DTr4Ed.

Tianbao Xie, Mengqi Yuan, Danyang Zhang, Xinzhuang Xiong, Zhennan Shen, Zilong Zhou, Xinyuan Wang, Yanxu Chen, Jiaqi Deng, Junda Chen, Bowen Wang, Haoyuan Wu, Jixuan Chen, Junli Wang, Dunjie Lu, Hao Hu, and Tao Yu. Introducing osworld-verified. *xlang.ai*, July 2025. URL https://xlang.ai/blog/osworld-verified.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, Jinrui Ding, Xiandi Ma, Yuchen Xie, Peng Pei, Xunliang Cai, and Xipeng Qiu. Evocua: Evolving computer use agents via learning from scalable synthetic experience, 2026a. URL https://arxiv.org/abs/2601.15876.

Tianci Xue, Zeyi Liao, Tianneng Shi, Zilu Wang, Kai Zhang, Dawn Song, Yu Su, and Huan Sun. Autonomous continual learning for environment adaptation of computer-use agents, 2026b. URL https://arxiv.org/abs/2602.10356.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

Limitations and Broader Impacts

VeriCUA trades cost for evidence. A verification pass issues tens of tool calls and model calls per trajectory, an order of magnitude more than a single-call judge, so it suits filtering and reward

assignment better than interactive use. On recorded trajectories it abstains wherever the recording holds no evidence for a requirement. Those abstentions are a coverage limit, and online access lifts it only for environments that can be instrumented. A stochastic verifier also needs its draw-to-draw spread reported beside every comparison, as we do. Reliable trajectory verification lowers the cost of training computer-using agents and of auditing benchmark labels. The same machinery could be misused to certify agents that act on real systems, which is one reason VeriCUA never writes to the environment it verifies.

A Environment Details

Action Space. Table 5 lists the verbs by state source and modality, together with the evidence type each one stamps.

Table 5: The action space by state source and modality, with the evidence type each verb stamps.

State Source	Modality	Verbs	Evidence Type	Class
actions and reasoning	text	`find_steps`, `inspect_action`	agent claim	asserted
	code	`compute`	sandboxed computation	attested
screen frames	GUI	`overview`, `get_frame`, `zoom`, `ocr`	post-effect frame; precise read	attested
			pre-action frame	asserted
a11ytrees	structured	`read_a11y`, `search_a11y`	structured state	attested
machine	code	`file_exists`, `find`, `list_dir`, `file_stat`, `read_file`, `sqlite_query`, `content_query`	environment read (persistent)	attested
	code	`dumpsys`, `current_activity`, `settings_get`, `getprop`, `pm_list`	environment read (persistent or volatile)	attested
	GUI, structured	`screenshot`, `ui_dump`	screenshot; UI dump (volatile)	attested
	text	`submitted_answer`, `clipboard_read`	harness artifact	attested

The Two Settings. Table 6 lists what each setting determines at each stage of the procedure.

Table 6: What each setting determines in the shared procedure.

Stage	Aspect	Online environment	Recorded trajectory
Map	where state lives	packages, stores, and schemas of the machine	steps and frames that concern each requirement
Explore	how actions are spent	synthesized checkers, each a getter plan plus one predicate	K audit paths, forward support and backward refutation
Stamp	attested types	environment read, UI dump, harness artifact	post-effect frame, precise read, computation, structured state
	setting-specific label	persistence, persistent or volatile	frame timing, before or after the action
Arbitrate	requirement met when	one persistent attested read confirms it	every path attests it
	unsettled requirements	folded to FAIL, or reported as abstention	resolved by the sealed initial judgment

Extension Contract. A new source enters as exactly (i) its read verbs, (ii) its evidence types with their attested or asserted classification, and (iii) an initial view for the cold-start judgment; every later stage consumes types, so nothing downstream changes. The accessibility channel entered this way, and the machine channel repeated the pattern with its own inventory.

Read-Only Enforcement. On recordings the read-only property is structural, since every verb is a pure function of files on disk. On a machine the verbs come from a fixed allowlist, arguments that could smuggle shell control are rejected, and a digest of the plan's persistent targets is compared before and after execution.

Online Android Harness. In online environments, VeriCUA runs against AndroidWorld on an Android emulator. The harness has root access over `adb`, so every store the benchmark's own evaluator can read is readable by the verifier. Each episode is executed by the benchmark's agent scaffold; when it ends, the harness saves the device state as a snapshot and verification runs against that snapshot, so any verification pass can be replayed on the identical end state. Before the episode starts, the harness records a pre-state anchor of the application storage, against which requirements about a change are compared. Verification actions come from a fixed allowlist of read-only getters over settings, files, SQLite stores, content providers, system services, the UI tree, the screen, and the submitted answer. Arguments that could carry shell control are rejected, and a digest of the plan's persistent targets is compared before and after execution. The agent's submitted answer is extracted by the harness and exposed through the `submitted_answer` getter as the artifact under test. The emulator is a headless Pixel 6 image at API level 33 (Android 13) with root access over `adb`. Snapshots are taken through the emulator console once the official evaluator has run, and a load restores the end state in about fifteen seconds. The multi-pass suite keeps every snapshot on which the agent claimed completion but gold says FAIL, on which the configurations disagree, or on which any pass abstained. One verifier pass takes about 100 seconds per episode, of which checker synthesis takes about half and predicate evaluation about a third, with getter execution and the scout pass under ten seconds together.

Recorded Trajectory Inputs. A recorded trajectory enters as the agent's actions and reasoning, its screen frames, and, when the platform recorded them, its a11ytrees. Frame semantics, whether frame i captures the state before or after action i, is declared per dataset and determines the post-effect and pre-action evidence types. The a11ytrees are condensed before they are read. OSReward trajectories are pure GUI with pre-action frame semantics; WebArena trajectories carry a11ytrees. The condenser writes one line per node of the browser's accessibility dump, indented by depth, with the node id, role, name, and ten retained properties such as value, checked, focused, and url. Ignored nodes and nameless nodes without properties are dropped, and names are cut to 110 characters. A read returns at most 150 lines with a note on how many nodes were left out, a query returns at most 40 matching lines, and a search across steps returns the first match in each of up to 12 steps. Each trajectory enters as one record that lists, per step, the agent's thought and action, the page URL, and the paths of the screenshot and the tree.

Other Online Environments. OSWorld runs in a virtual machine and ScienceBoard in its own scientific desktop environment. Both are verified through the same allowlisted read-only getters over the machine, and ScienceBoard's evaluator reads the state of the scientific applications inside the VM. Both use the virtual machine images that the benchmarks ship, without modification.

B Experimental Details

Datasets. Table 7 lists every evaluation set in the paper.

Baseline Implementations. TRM scores the trajectory from 1 to 5 in a single call from the instruction, every thought and action, and the last three frames; CUAJudge checks up to six task-derived key points against the first and last of up to six frames in a single call; OS-Themis verifies up to four ordered milestones over up to eight frames and reviews them in a second call. The online VLM judge answers in one word from the instruction and the final screenshot and never abstains; the auto-generated verifier writes one evaluator script per task with full device access and runs it

Table 7: Evaluation sets.

Set	Size	Gold	Notes
OSReward-Hard	284	human labels	hard subset of OSReward
OSReward full	1019	human labels	
WebArena	715	official checkers	239 / 235 / 241 rollouts by Claude Sonnet 4.6, Gemini 3.1 Pro, Gemini 3 Flash
AndroidWorld	116	official evaluator	M3A Agents
OSWorld, online	361	official evaluator	one rollout per task; actor success 44.3%
ScienceBoard, online	169	official evaluator	actor success 23.7%

once. The three recorded-trajectory judges run under one harness at temperature 0, with images downsampled to a long side of 1280 pixels, the first and last frames pinned, no self-consistency voting, and a parse failure mapped to FAIL. TRM thresholds its 1–5 score at 4 and is the only judge that sees the agent's thoughts. CUAJudge collapses the original four-stage pipeline into one call, dropping the dedicated frame-selection and init-versus-final modules. OS-Themis collapses the multi-agent critic to two calls with the budget cut to four milestones and eight frames against the original six and twelve; Table 8 shows that restoring the original budget changes little. The auto-generated verifier receives the instruction with its concrete parameters, the same generic device context our synthesis receives, unrestricted host-side `adb` and Python, and the agent's submitted answer on answer-scored tasks. It does not receive the trajectory, the scout's map, or the pre-state anchor, and it runs last on each end state. Majority voting combines the same three verifier draws by vote with abstentions folded to FAIL.

Table 8: OS-Themis budget ablation, mean balanced accuracy.

Configuration	BalAcc
OS-Themis, 4 milestones / 8 frames (used in the main text)	74.2
OS-Themis, 6 milestones / 12 frames (original budget)	75.9
CUAJudge	76.8

VeriCUA Configuration. On recorded trajectories, K audit paths each gather support forward and hunt refutations backward. A requirement is met only by a full vote of attested evidence across paths and unmet only with corroboration across independent probes, and anything unsettled is resolved by the sealed initial judgment. The default runs $K = 3$ paths and stops after two when they agree on the status of every hard requirement, which happens on about two fifths of the trajectories. A path spends up to four tool calls gathering support and then at least four, or two per hard requirement, hunting refutations. Each call returns one observation, frames are downscaled to at most 1288 pixels on the long side, and only the last eight images stay in context. The rubric comes from three candidate decompositions consolidated in a fourth call. The sealed initial judgment reads the last five frames together with the thought and action log and is reused only as the fallback. In online environments, a scout pass maps the device's packages, stores, and schemas before synthesis; each requirement receives checkers on more than one surface where possible; predicates are evaluated by a restricted evaluator or, for semantic comparisons, by a judge call; a requirement is met by one persistent attested confirmation and unmet by two corroborating refutations (the `unmet_min` threshold, which trades decided episodes for precision when raised). Across passes, the combination rule takes the decided verdict and abstains when two decided verdicts disagree. Rubric synthesis draws two candidate rubrics at temperature 0.7 and consolidates them in a third call at temperature 0; checker synthesis makes two parallel calls at temperature 0.7 and asks for two checkers per hard requirement on different surfaces where the state allows it; the scout pass is one call over at most three packages and six databases, followed by one schema probe; a semantic predicate costs one judge call at temperature 0 over at most three images. No call carries a token cap, and each has a 180-second timeout with up to six retries. The getter inventory has seventeen verbs, listed below with their persistence tags.

The two image verbs are enabled only when the vision fallback is on, `clipboard_read` runs last because it disturbs the screen, and a miss from `find` counts only as weak evidence.

persistent	`settings_get`, `getprop`, `file_exists`, `file_stat`, `read_file`, `list_dir`, `find`, `sqlite_query`, `content_query`, `pm_list`, `view_file_image`
volatile	`dumpsys`, `ui_dump`, `current_activity`, `clipboard_read`, `submitted_answer`, `screencap` (the screenshot verb of Table 5)

Cost. Table 9 reports tokens per trajectory or episode.

Table 9: Cost per trajectory or episode.

Setting	Method	Tokens
recorded, WebArena (tree-fed, flash)	three judges together (4 calls)	43K–61K
recorded, WebArena (GUI, pro)	three judges together (4 calls)	25K
recorded, OSReward (GUI)	three judges together (4 calls)	33K–35K
recorded, WebArena	VeriCUA	150K–208K
recorded, OSReward	VeriCUA	345K–378K
online, first suite	auto-generated verifier	16.2K (60 s)
online, first suite	VeriCUA	39.2K (97 s)
online, multi-pass suite	auto-generated verifier	21K
online, multi-pass suite	VeriCUA, per pass	43K–51K

C Additional Results

Multi-Pass Combination In The Online Environment. We also tested on a multi-pass AndroidWorld suite to test stability; five verifier passes share one restore per episode: three independent draws of one configuration, one draw with a vision fallback, and the auto-generated verifier. Table 10 reports them singly and combined. Adding identical draws buys little, whereas adding the pass that is blind elsewhere buys most of the gain, and our combination rule keeps wrong-FAIL verdicts an order of magnitude below majority voting over the same draws.

Table 10: Multi-pass combination in the online environment.

Pass	Agreement	Decision Rate	Decided Acc	False Positives	Wrong FAILs
draw 1	75.7	72.2	91.6	3	4
draw 2	79.1	77.4	92.1	3	4
draw 3	77.4	67.0	94.8	1	3
draw with vision fallback	76.5	73.0	90.5	3	5
three draws combined	80.9	78.3	95.6	2	2
three draws + vision fallback	**87.0**	**87.0**	**96.0**	2	2
draws 1, 2 + vision (equal cost)	84.3	85.2	94.9	3	2
three draws, majority vote	80.0	100	80.0	3	20
auto-generated verifier	78.3	100	78.3	4	21

Single-Call Judges On Other Backbones. Table 11 lists the single-call judge of OSReward on several backbones beside VeriCUA on `gemini-3-flash-preview`.

Backbone Study In The Online Environment. Table 12 repeats the flash-to-pro comparison of Table 4 in the online AndroidWorld environment.

Table 11: The OSReward single-call judge by backbone, with VeriCUA for reference.

	OSReward-Hard		OSReward full	
Judge	Acc	BalAcc	Acc	BalAcc
VeriCUA (Gemini 3 Flash)	**70.4**	68.3	87.2	86.3
Gemini 3 Flash	57.0	65.2	87.0	88.2
Gemini 3.1 Pro	61.6	61.6	87.9	88.2
Claude Sonnet 4.6	59.2	68.1	87.7	88.9
Claude Opus 4.6	67.3	68.6	89.5	90.2
Claude Opus 4.8	69.7	**69.7**	**89.7**	90.0
GPT-5.5	67.3	67.0	89.5	89.8

Table 12: Accuracy and balanced accuracy in the online AndroidWorld environment with each framework on the flash and pro tiers.

Judge	**Acc**			**BalAcc**		
	Flash	**Pro**	Δ	**Flash**	**Pro**	Δ
VLM judge	78.4	79.7	+1.3	77.5	78.5	+1.0
Auto-generated verifier	82.4	85.1	+2.7	82.7	85.4	+2.7
VeriCUA	**90.5**	**91.9**	+1.4	**90.3**	**91.8**	+1.5

Evidence Rules And Recall. Figure 6 shows which arbiter rules carry the result on the first AndroidWorld suite and how episode recall falls with the number of hard requirements.

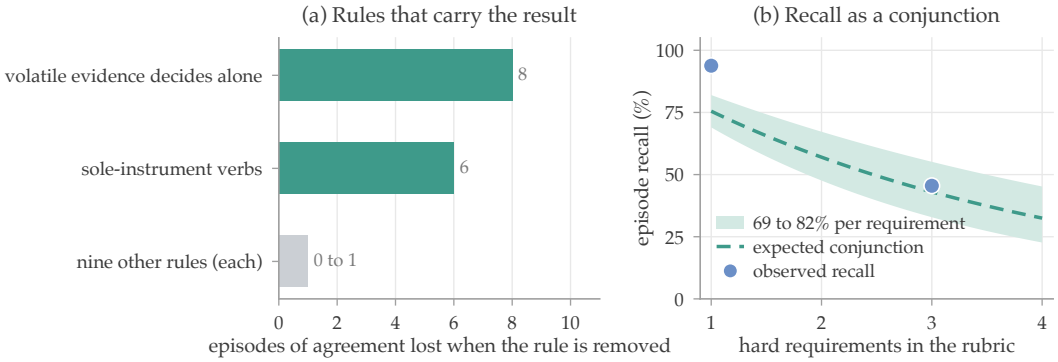

Figure 6: Evidence rules and recall on the first AndroidWorld suite.

Action-Space Ablation. The values behind Figure 3 are in Table 13.

Table 13: Action-space ablation in the online AndroidWorld environment.

Action Space	**Prec**	**Rec**	**F1**	**Acc**	**False Positives**	**Decided**	**Decided Acc**
GUI only	86.4	57.6	69.1	77.0	3/41	45%	87.9%
Hybrid (default)	**90.6**	**87.9**	**89.2**	**90.5**	3/41	82%	**95.1%**

VeriCUA By Observation Input. Table 14 gives VeriCUA's accuracy, decision rate, share of verdicts resting on structured evidence, and cost under each observation input on WebArena.

WebArena Per Rollout Agent. Table 15 breaks the pooled WebArena results down by rollout agent.

Table 14: VeriCUA by observation input on WebArena.

Input	Acc	BalAcc	Decided	Structured Evidence	Tokens
Screenshots	80.0	80.0	33%	0%	234K
Screenshots + a11ytrees	80.0	80.0	75%	73%	252K
a11ytrees	76.7	76.7	85%	93%	178K

Table 15: Accuracy and balanced accuracy on WebArena per rollout agent and pooled.

	Rollout Agent						Pooled	
	Claude Sonnet 4.6		Gemini 3.1 Pro		Gemini 3 Flash			
	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc
VeriCUA (GUI + a11ytree)	**79.9**	**80.3**	**84.3**	**85.3**	**83.8**	**83.9**	**82.7**	**83.1**
CUAJudge	79.1	79.8	82.1	83.4	82.6	81.9	81.3	81.8
TRM	76.2	77.0	79.1	80.8	83.4	81.0	79.6	79.8
OS-Themis	70.3	69.8	80.0	79.4	75.9	74.1	75.4	74.3

Tree-Fed Baselines On WebArena. The judges are re-run with a11ytrees alone as their input, in place of the GUI and a11ytree input of Table 15, per rollout agent and pooled (Table 16).

Table 16: Accuracy and balanced accuracy on WebArena with the judges reading a11ytrees.

	Claude Sonnet 4.6		Gemini 3.1 Pro		Gemini 3 Flash		Pooled	
	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc
VeriCUA (GUI + a11ytree)	**79.9**	**80.3**	**84.3**	**85.3**	**83.8**	**83.9**	**82.7**	**83.1**
CUAJudge, tree-fed	77.8	77.9	83.0	84.3	82.6	80.0	81.1	80.8
TRM, tree-fed	74.5	74.9	80.4	81.4	80.9	77.4	78.6	78.1
OS-Themis, tree-fed	74.5	73.5	79.6	78.7	78.0	74.9	77.3	75.7

Confident Failures And Gold-Label Errors. Table 17 gives the size and precision of the confident-failure tier, the trajectories on which VeriCUA decides FAIL and its sealed initial judgment agrees, on each of the three rulers.

Table 17: Precision of the confident-failure tier by ruler.

Ruler	Tier Size	Precision
WebArena, official checkers	212	92.0
OSReward-Hard, human labels	54	88.9
OSReward full, human labels	179	92.7

The gold-label audit starts from the 814 recorded trajectories of AndroidWorld, OSWorld, and WebArena on which VeriCUA's typed rule decides. A trajectory is flagged when that decision contradicts the benchmark's gold label, which happens on 81 of them. A single-blind adjudicator, a model from a different family than the verifier's backbone, then re-judges each flagged trajectory from scratch. It sees only the instruction, the agent's thought and action log, and up to ten evenly spaced frames including the last one, never the verifier's verdict, the gold label, or the verifier's reasoning, and it may answer SUCCESS, FAIL, or UNSURE. A flag counts as confirmed when the adjudicator sides with VeriCUA against gold. Of the 81 flags, 51 are confirmed, 20 refuted, and 10 left UNSURE, with UNSURE kept in the denominator. The rate holds only on OSWorld (37 of 57), where a second adjudicator from a third model family, run under the same blind protocol, confirms

34 of the 51 flags it decides. In the online OSWorld environment, about 18 of VeriCUA's 28 false positives are episodes that the official evaluator marks FAIL in error.

D CASE STUDIES

Each card shows the instruction and the frames or machine reads the verifier cited. The summary below it gives the verdict against gold and the baselines, the rubric with each requirement's final status, and the typed evidence the arbiter used.

Online Environments. The verifier reads the live machine after the episode ends.

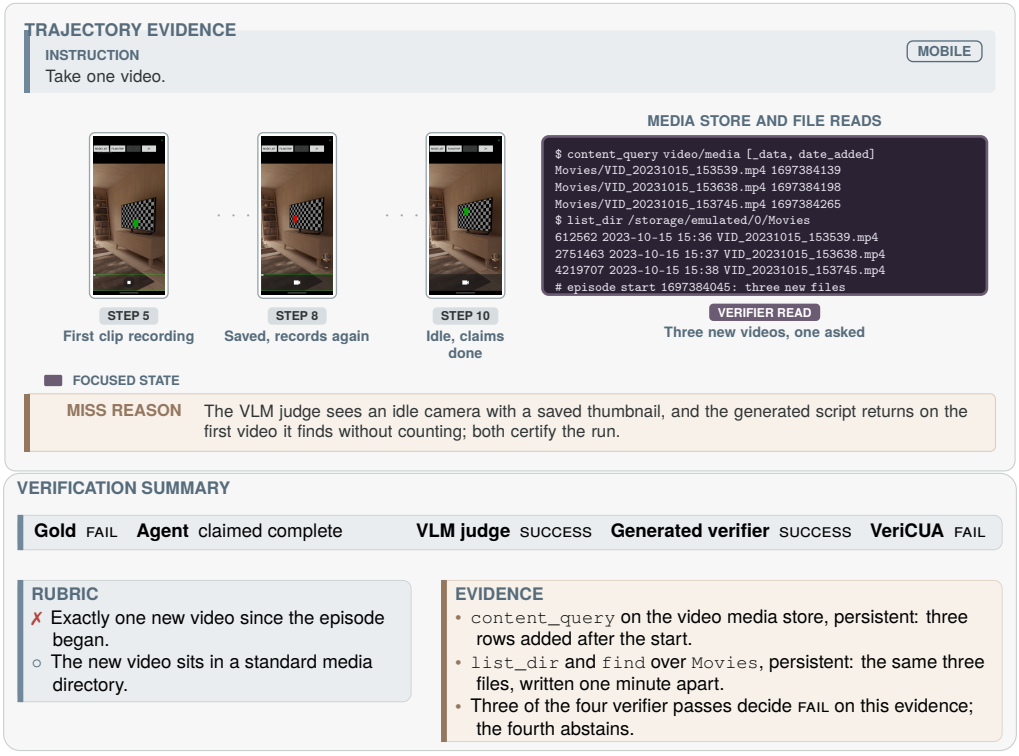

Figure 7: A false completion claim refuted from the media store. The agent recorded three clips for one requested video, and both baselines certified the run.

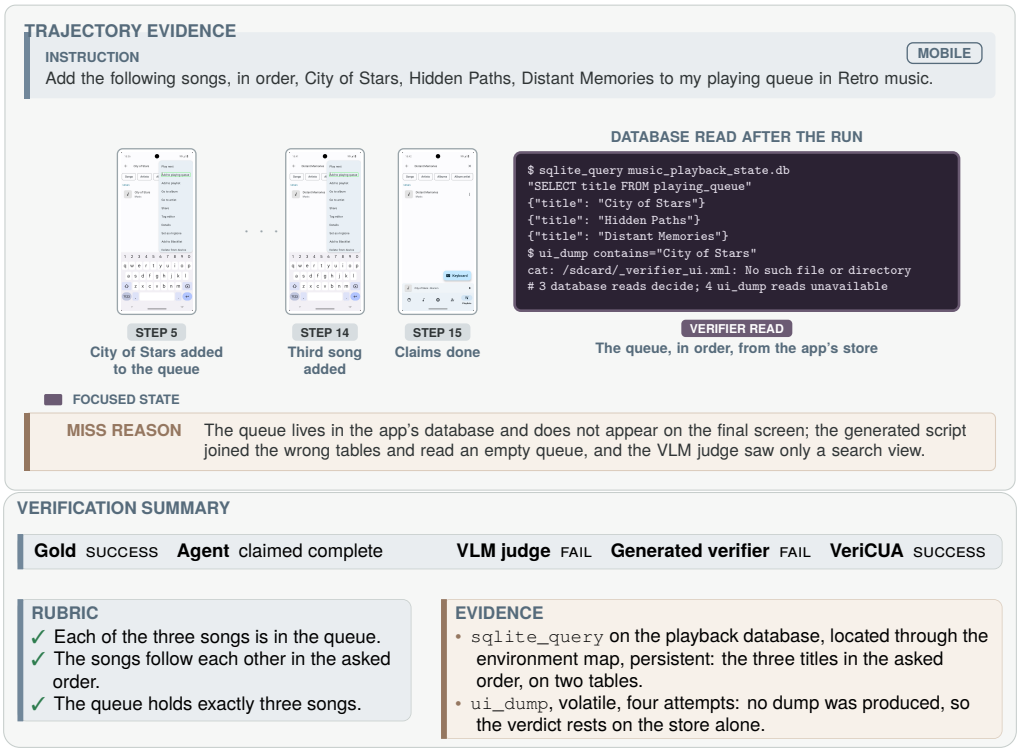

Figure 8: A success that only the app's store can attest. The queue is read from the playback database in the asked order, while both baselines fail the run.

Recorded Trajectories. Here the verifier has nothing but the frames and logs the recording kept.

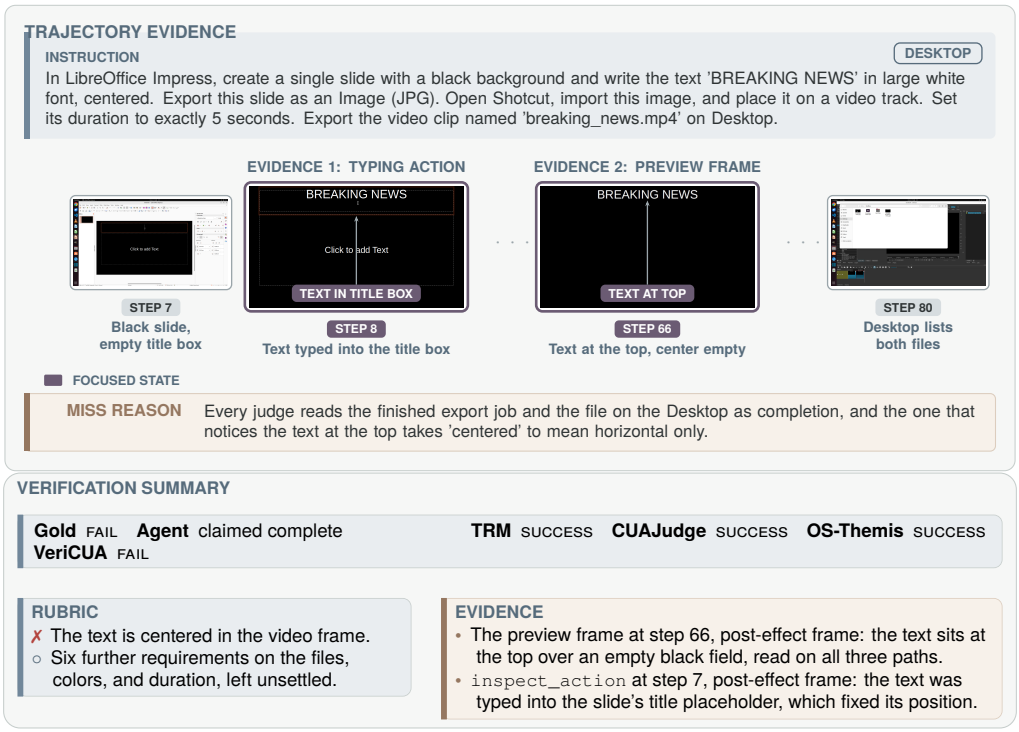

Figure 9: A long desktop recording that every judge passes on the finished export. The verifier reads the preview frame and finds the text at the top rather than centered.

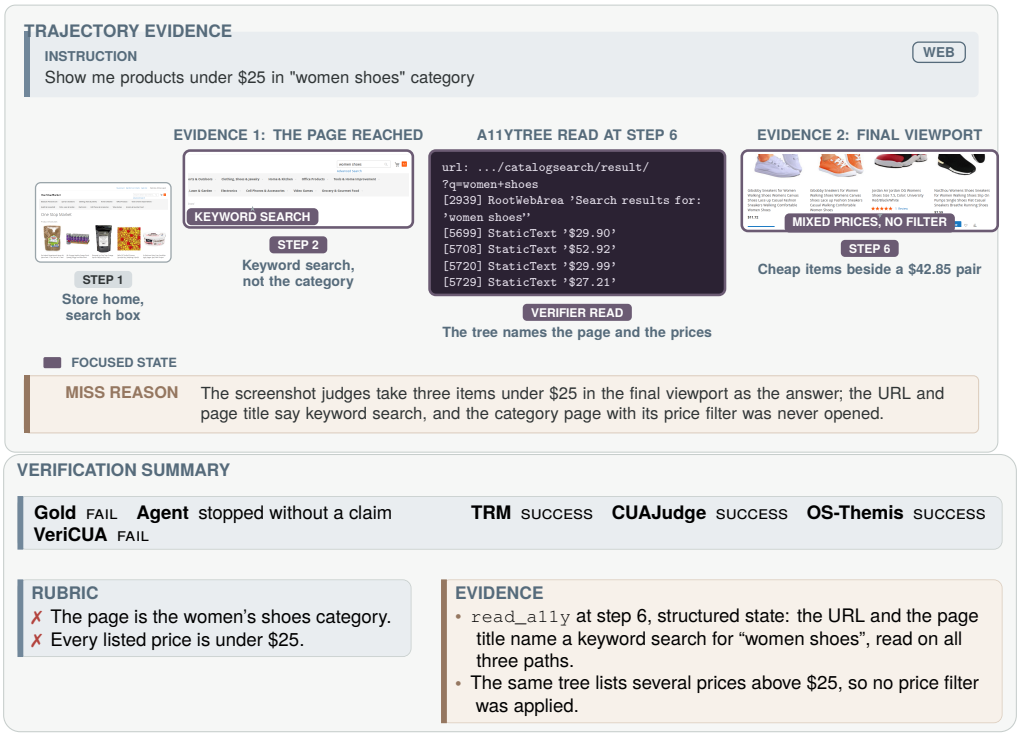

Figure 10: A web trajectory whose final screen shows cheap shoes. The a11ytree names the page a keyword search rather than the category page and lists prices above the limit.

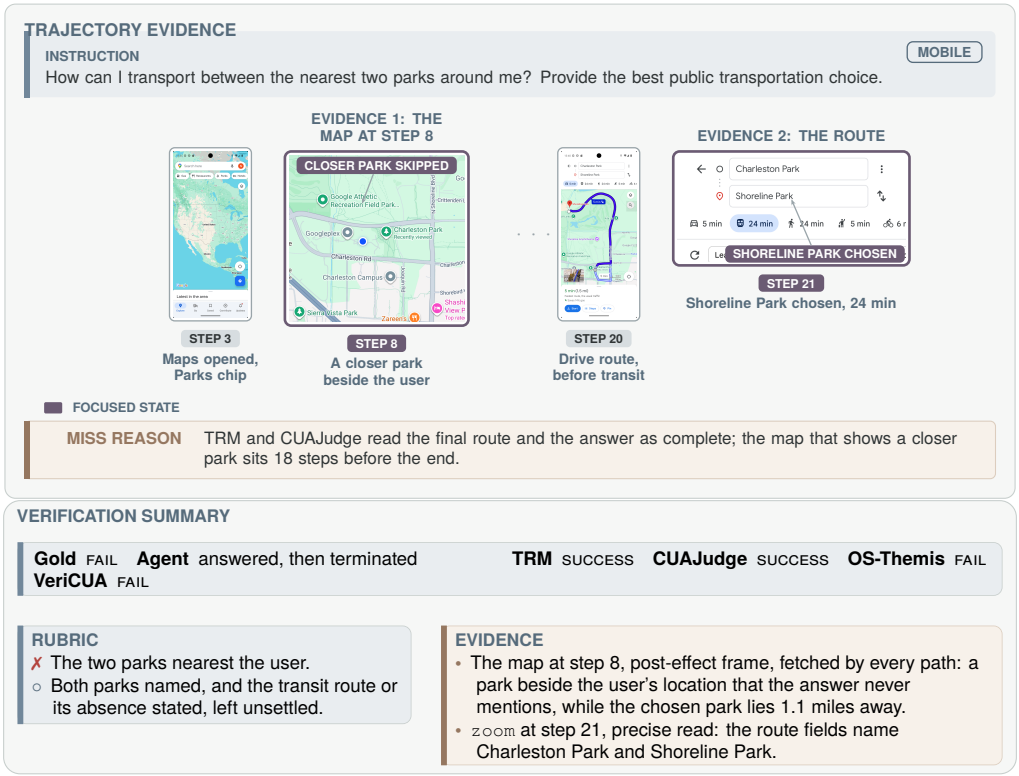

Figure 11: A mobile recording whose answer reads well. The map 18 steps before the end shows a closer park than the one the agent chose, and two of three judges passed it.

E ACCESSING VERICUA

VeriCUA will be released as two packages, for recorded trajectories and for online environments. Each package ships with its harness, getter allowlist, and baseline implementations. The release location and setup instructions will be provided upon publication.

F PROMPTS

The system prompts below drive rubric synthesis in the two settings and checker synthesis in online environments. Task-specific inputs such as the instruction and the device context are appended at run time. The audit and probe prompts of the recorded setting follow the same form.

```
You decompose a computer-use task into atomic, individually checkable
requirements for a verifier. Output requirements about the FINAL END-STATE the
task demands (what a programmatic checker inspecting files/documents/settings
afterwards would verify), NOT about the method or the steps taken.

Rules:
- Each requirement is one concrete, verifiable condition.
- Honor every quantifier/detail: "all"/"each" => ALL; exact names/values/paths
  => exact. Split compound asks into separate requirements.
- Mark hard (must hold for success) or soft (minor/cosmetic).
- Do NOT invent process requirements ("must have clicked X"); if the end-state
  is right, the route does not matter.

Output ONLY:
```json
{"requirements": [{"id": "r1", "description": "...", "hard": true}, ...]}
```
```

Prompt 1: Rubric synthesis on recorded trajectories.

```
You author verification requirements for tasks executed on an Android device.
Given a task instruction, produce the END-STATE requirements a strict human
examiner would check AFTER the task supposedly finished.

Rules:
- END-STATE only: describe what must be true of the device state, never the
  procedure ("brightness setting equals maximum", not "user opened settings").
- Atomic: one checkable fact per requirement. Split name vs content vs location.
- Strict reading: if the instruction names a value, place, or format, it is
  required exactly; include implicit artifacts (a created note must exist AND
  contain the given text).
- hard=true for requirements whose failure fails the task; hard=false for
  cosmetic/soft ones.
- Only what the instruction implies: do NOT add adjacent settings, modes, or
  hygiene the instruction never mentions (no invented requirements).
- COUNTS ARE EXACT: "take one video" means exactly one new video was created,
  not at least one; "log the reimbursable transactions" means exactly that set
  was added -- nothing missing, nothing extra. When the instruction quantifies,
  produce a requirement for the exact count or exact set of NEW artifacts
  (relative to the episode start), not mere existence.
- Do NOT bind platform-specific encodings or exact constants the instruction
  does not state: say "the minimum value the system allows", never "(0)".
  Resolving concrete encodings is the checkers' job, not yours.
- Do NOT bind storage locations or app conventions the instruction does not
  name ("a photo was taken" requires a new image in SOME standard media
  location, not specifically DCIM). Location is a requirement only when the
  instruction states one.
Output ONLY JSON:
```json
{"requirements": [{"id": "r1", "description": "...", "hard": true}]}
```
```

Prompt 2: Rubric synthesis in online environments.

```
You write CHECKERS that verify end-state requirements on an Android device.
A checker is DATA, not a script: a plan of read-only getter calls plus one
predicate over their outputs. You cannot write to the device; only these
getters exist:

{getter_docs}

{device_context}

Design rules:
- For each hard requirement produce 2 checkers probing DIFFERENT surfaces or
  routes where meaningful (e.g. settings vs dumpsys; direct path vs find).
  If only one sensible surface exists, 2 independent probes of it are fine.
- Prefer broad reads + predicate (list_dir or find + check the listing) over a
  single narrow existence probe: the task may have been satisfied at a
  different valid location or filename than the most literal guess.
- predicate kinds: "code" (one Python expression over the getter outputs)
  or "semantic" (a plain-language criterion decided by a separate judge).
  [rules on polarity, exact-name matching, store representations, and
  screen-only state omitted for space]
```

Prompt 3: Checker synthesis in online environments, shortened where marked.