SEEKJUDGE: A PRACTICAL REWARD FRAMEWORK FOR REINFORCEMENT LEARNING IN COMPUTER-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deciding whether a trajectory actually fulfills its instruction determines how we measure computer-use agents on long-horizon graphical-user-interface tasks and how we train them with reinforcement learning (RL). This judgment has long relied on rule-based evaluation. Rule-based evaluation often disagrees with human intention and becomes outdated when an app updates or its online content drifts. Existing model-based judges attempt to address these problems, but their judging accuracy remains limited. We propose the **SeekJudge** framework, whose four agents, a Condense, a Ground, a Seek and an Analyze agent, reach a verdict through a Seek–Analyze loop over the trajectory. To train a specialized model on densely labeled trajectories, we propose a seed-driven distillation pipeline that expands a few human-labeled seed trajectories into 10K such trajectories. To evaluate step-level judgments, we build CUAStepBench, a human-annotated benchmark that pairs trajectory verdicts with dense step labels. Beyond accuracy, SeekJudge costs far less than a closed-source large model. We further propose rollout overlap, a reward-server design that reduces the overhead of a reward model in RL training. To our knowledge, SeekJudge is the first model-based reward to match or surpass native rule-based reward in online RL, measured by downstream success rate on held-out RL test goals. In offline judging, SeekJudge-9B also exceeds rule-based evaluation by 9.5 F1 on AgentRewardBench.

1 INTRODUCTION

Computer-use agents (CUAs) have moved from constrained synthetic environments such as MiniWoB (Shi et al., 2017) to open-ended tasks across web, mobile and desktop platforms (Zhou et al., 2024; Deng et al., 2023; Xie et al., 2024; Rawles et al., 2025), and their trajectories have grown from tens of steps to hundreds of steps (Yuan et al., 2026). Deciding whether the long trajectory an actor produces fulfills the instruction determines how we measure a model and how well downstream reinforcement learning or rejection fine-tuning works.

For a long time, judging a trajectory has relied mainly on rule-based evaluation (Xie et al., 2024; Rawles et al., 2025; Zhou et al., 2024; Wang et al., 2026). Rule-based evaluation suffers from three problems. **(1) A rule disagrees with human intention.** Many valid answers can satisfy the same goal, yet a rule cannot enumerate them all (Zhu et al., 2025). The extreme case is an open-ended instruction such as drawing a picture, where acceptable outputs are unbounded and no rule can judge the result. On an existing CUA benchmark, a blind human re-check of every trajectory where rule-based evaluation and a Codex judge give opposite verdicts agrees with the judge more often than with the rule, as Figure 1(a) shows. **(2) A rule needs a dedicated parser per application and breaks across application versions.** A rule reads internal state through layout or internal APIs, so each new application needs its own parser or a mock app, and the parser stops working once the application moves to a new version. **(3) A rule's ground-truth answer becomes outdated.** Even when the application stays the same, the content of a live website keeps changing. A rule relies on a predefined answer, so once that content changes the answer drifts or the task becomes unsolvable, for example when a task asks for the CEO of a company and the company later appoints a new one. Covering live online services therefore incurs a heavy maintenance cost (Xue et al., 2025).

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.

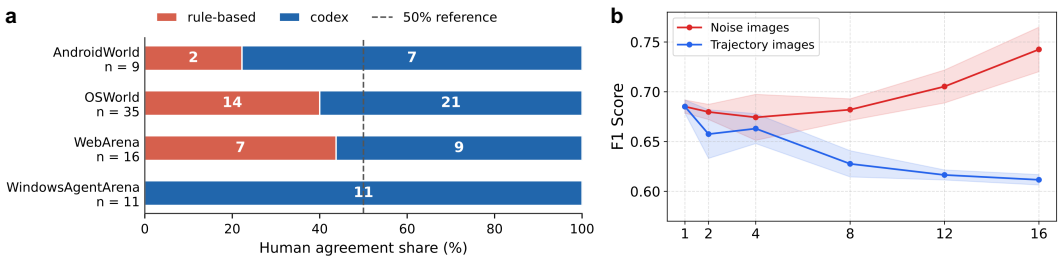


Figure 1: **(a)** A Codex judge runs over about 1K trajectories of OmniGUIRewardBench. On the 71 trajectories where it and rule-based evaluation disagree, a blind human re-check agrees with the judge (*blue*) more often than with the rule (*orange*). **(b)** With the decisive screenshot always present, padding with other trajectory images (*blue*) lowers F1 while equal-token noise images (*red*) do not. Details are in Appendix C.

These problems of rules motivate a shift from rule-based to model-based evaluation, but existing frameworks have not yet made this paradigm mainstream, because their judging accuracy remains limited (Xue et al., 2026; Li et al., 2026; Rosset et al., 2026). These frameworks put many images into one context while only a small high-fidelity fragment determines the verdict, and thus they may overlook the decisive evidence. Even though the decisive image is always present, performance degrades increasingly with the number of trajectory images added to the same forward pass, as Figure 1(b) shows.

We decompose judging a trajectory into two subtasks: localization, which identifies the image holding the decisive evidence, and extraction, which reads the decisive detail from that image accurately. Based on this decomposition, we propose **SeekJudge**, a multi-agent model-based judge that separates the two subtasks. Its Seek agent localizes on a text timeline, and its Analyze agent extracts the decisive detail from one image at a time. Its Condense agent records the state transition of each step and its Ground agent identifies the element that each action hits, which together build this timeline from the trajectory. SeekJudge also has the fine-grained scoring, low cost and low latency that a reward model inside RL needs:

- **On judging quality**, SeekJudge returns a continuous score for the trajectory and classifies every step, which gives a graded reward and step-level judgments.
- **On efficiency**, the whole system is a single 9B open-source model, hundreds of times cheaper than a closed-source large model, and its peak context stays 4–6$\times$ smaller than that of existing model-based judges. Rollout overlap further runs part of the judging while the environment executes actions, so the reward model adds less time to the rollout.

In summary, our contributions are as follows. **(1)** We propose SeekJudge, a multi-agent framework that separates localization from extraction, together with rollout overlap that makes it faster inside reinforcement learning. **(2)** We construct CUAStepBench, a CUA reward benchmark that pairs human trajectory verdicts with dense step labels, spanning 177 applications across platforms. **(3)** We train SeekJudge-9B for the framework. To our knowledge, it is the first model-based reward to match or surpass native rule-based reward on downstream test success in online RL. It also exceeds existing model-based judges on offline reward benchmarks.

2 Related Work

From rule-based to model-based reward. Deciding whether a trajectory fulfills its instruction underlies both benchmark scores and the RL reward. Both mainly rely on rule-based evaluation (Xie et al., 2024; Rawles et al., 2025; Zhou et al., 2024; Wang et al., 2026). These rules can themselves be wrong, and OSWorld has fixed its evaluators after users reported errors (Xie et al., 2025). Model-based judges replace the rule with a model. CUAJudge (Xue et al., 2026) judges whole trajectories in multi-image forward passes, where the decisive screenshot competes with many others, OS-Themis (Li et al., 2026) runs a multi-agent critic whose context grows with trajectory length, and ProRe (Dai et al., 2026) and VAGEN (Cui et al., 2026) probe the live environment after the episode ends. As RL

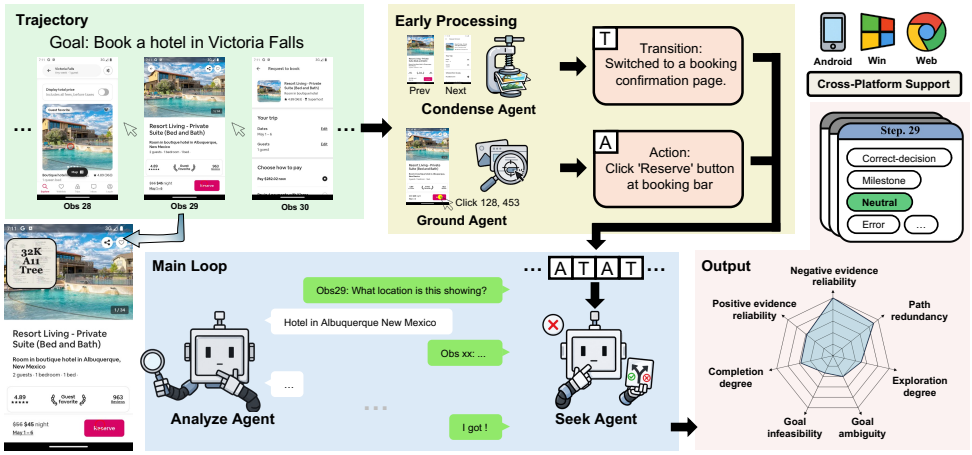


Figure 2: The SeekJudge framework. Four agents share one backbone. Condense and Ground write a text timeline with one transition entry (T) and one grounded-action entry (A) per step.

rewards, OpenWebRL-Judge (Yang et al., 2026), OS-Themis and VAGEN are compared only with other model-based rewards or the untrained policy, WebJudge (Xue et al., 2025) and ProRe leave RL to future work, and StainFlow (Hao et al., 2026) and ADMIRE (Zheng et al., 2026) add their rewards to the rule-based reward of AndroidWorld rather than replacing it. PRO-CUA (He et al., 2026) outperforms a rule-based baseline in web RL, but that rule matches reference actions and a GPT-5 judge scores success. To our knowledge, this paper is the first to train with a model-based reward in place of the native rule-based reward and compare the two.

Step-level reward models. A trajectory verdict does not say which steps went wrong. Process reward models score each step in mathematics (Lightman et al., 2024; Wang et al., 2024) and web navigation (Chae et al., 2025). For CUAs, the World-State-Model judge of SEAgent (Sun et al., 2026b) covers only narrow Chrome tasks, GUI-Owl (Ye et al., 2025) embeds step critics inside the actor pipeline, and OpenCUA (Wang et al., 2025b) sees only the context up to each step, while VLM rewards for GUI RL (Yang et al., 2025; Qi et al., 2025; Bai et al., 2024; Sumyk & Kosovan, 2026) and trajectory filtering (Pan et al., 2024; He et al., 2024) are generic judges (Zheng et al., 2023) without step labels. SeekJudge classifies every step from a timeline of the whole trajectory.

Reward benchmarks. Evaluating step judgments needs step labels, which AgentRewardBench (Lù et al., 2025) and OmniGUIRewardBench (Li et al., 2026) lack. CUARewardBench (Lin et al., 2026) labels sparse key actions on Ubuntu only, and OS-Critic Bench (Wu et al., 2026) labels unexecuted candidate actions without a trajectory verdict. CUAStepBench labels every executed step together with the trajectory verdict.

3 METHOD

3.1 THE SEEKJUDGE FRAMEWORK

SeekJudge realizes localization and extraction as a multi-agent framework whose four agents share one backbone model (Figure 2). The Condense and Ground agents first compress the whole trajectory into a compact text timeline. The Seek agent takes this timeline as its initial context and runs a Seek–Analyze loop, querying the Analyze agent for the evidence that it needs before giving its judgment.

Seek agent. The Seek agent controls the loop and is the only agent that keeps state across the rounds of one judgment. When the condensed timeline holds enough evidence for a verdict, it outputs the seven dimension scores of the trajectory and a step class for each step. When the timeline lacks that evidence, it selects one screenshot, sends a focused question about it to the Analyze agent, and appends the answer to its context. The loop runs for at most 8 rounds.

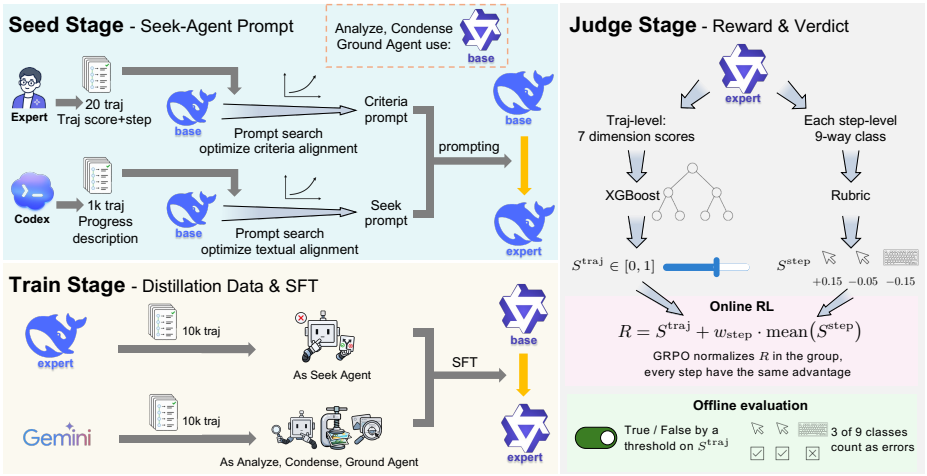


Figure 3: The seed-driven construction of training data, the evaluation process, and the reduction of the dense labels to a trajectory verdict and step labels for offline evaluation and to one reward per rollout for online RL (Eq. 1).

Analyze agent. The Analyze agent answers one focused question about the single screenshot that the Seek agent selects and returns the decisive detail as text. Each answer sees only that question and image.

Condense and Ground agents. For each step the Condense agent reads the screenshots before and after the action and records the state transition in a few lines of text as the T entry of the timeline. The Ground agent reads the screenshot before the action together with the raw action that the actor executed, such as `click 128,453`, identifies the element that those coordinates hit, and records it as the A entry of the timeline.

Separating localization from extraction gives three properties. **(1)** Every agent reads at most two images at a time, at high fidelity, with one screenshot for Analyze and Ground and two for Condense. Appendix C shows that, even with the decisive screenshot always in the input, a verdict degrades as more screenshots of the same trajectory share one forward pass. **(2)** Evaluation scales to longer trajectories because Condense and Ground compress every step into a few lines of text, and Section 4.5 measures a peak context 4–6$\times$ smaller than that of existing judges. **(3)** Compute adapts to length because the Seek–Analyze loop issues only the tool calls to Analyze that a case needs, and Appendix D counts these tool calls against trajectory length.

3.2 Dense Supervision via Seed-Driven Distillation

To train the single 9B backbone that the four agents share, we build supervised data for each of the four roles through a seed-driven construction with a Seed and a Train Stage, and a Judge Stage reduces the dense outputs to scalars, as Figure 3 summarizes.

Dense label design. A single true/false label is coarse in two ways. It cannot rank several trajectories of one task beyond pass or fail, and it merges the distinct axes of success or failure, such as the fraction of the task completed or the confidence of the judgment, into one bit. To separate these axes, we score each trajectory on seven dimensions, namely goal infeasibility, goal ambiguity, exploration degree, path redundancy, positive evidence reliability, negative evidence reliability, and completion progress, each on a 0–100 scale. We assign each step one of nine classes instead of a true/false mark.

Seed Stage. This stage tunes the two prompts that DeepSeek-V3.2 uses as the Seek agent. The criteria prompt defines what counts as success and is searched against about 20 trajectories that a human labels with seven dimension scores and step classes. The seek prompt specifies what to ask the Analyze agent and when to stop and is searched against the progress descriptions that Codex

with GPT-5.5 writes for about 1K trajectories. Both searches run in Claude Code, and Appendix E.1 describes how neither search reads benchmark ground truth.

Train Stage. We run the full SeekJudge pipeline with the Seed Stage prompts, DeepSeek-V3.2 as the Seek agent and Gemini-3.0-Flash as the Analyze, Condense and Ground agents, and record the input and output of every agent step as distillation data. Supervised fine-tuning (SFT) of Qwen3.5-9B on this data gives SeekJudge-9B. Appendix K.1 lists the SFT data and hyperparameters.

Judge Stage. The framework outputs the seven dimension scores of the trajectory and a step class for each step, and the Judge Stage reduces these scores and classes to the scalars that offline evaluation and online RL use. A small gradient-boosted regressor maps the seven dimension scores to a trajectory score S^{traj}, and each of the nine step classes maps to a step score S^{step} set by a human rubric. Offline evaluation and RL each use one fit of the regressor, and Appendix E.2 lists both fits and probes the offline fit for leakage. With the offline fit alone in place of the RL fit, SeekJudge still surpasses rule-based reward in RL on UI-TARS-1.5-7B Impress, as Appendix F shows.

- **For offline evaluation**, one threshold on S^{traj}, shared by every benchmark and backbone, gives the trajectory verdict, and the three harmful classes count as erroneous steps while the remaining six count as correct. Appendix L defines the nine classes, marks which count as correct, and lists their step scores.
- **For online RL**, the reward server returns, for rollout i of a group of G rollouts that share one goal, the trajectory score S_i^{traj} and the step scores $S_{i,1}^{\mathrm{step}}, \ldots, S_{i,N_i}^{\mathrm{step}}$ of its N_i steps. The trajectory reward adds the mean step score to the trajectory score with weight $w_{\mathrm{step}} = 2$,

$$R_i = S_i^{\mathrm{traj}} + \frac{w_{\mathrm{step}}}{N_i} \sum_{k=1}^{N_i} S_{i,k}^{\mathrm{step}}. \tag{1}$$

  GRPO normalizes R_i within the group and assigns the resulting advantage to every step of rollout i. This advantage carries no step index k, and the step scores enter it only through their mean in Eq. 1. This is the weak form of a process reward, with the step term assisting the trajectory-level reward, and it is the default throughout the paper. Appendix K.3 gives the implementation, and Appendix B tests the strong form, which injects the step scores into the advantage of every step.

Training data. We collect training trajectories for 10,905 distinct tasks across web, desktop (Windows, Ubuntu and macOS) and Android. Every task goal is a human-written description, and the set has zero overlap with any reward benchmark that we tested or any OSWorld goal. Appendix I gives the composition of the set by environment and application.

3.3 Rollout-Overlapped, Zero-Client-State Reward Server

Serving a reward model inside RL raises a scheduling problem. In a GUI rollout, the environment executes each action of the multi-turn loop while the actor's GPUs idle, and a judge that runs only after the trajectory ends adds its whole latency to every rollout. We design the reward server around this loop with two mechanisms. Rollout overlap fills the idle window with judging, and zero client state keeps the state management that this overlap needs out of the RL-side code.

Rollout overlap. We overlap judging with the rollout so the reward model adds less latency to training. Operations that do not depend on later actions, such as Condense and Ground here or key-information identification in CUAJudge, run as soon as their screenshots and action arrive, in the idle GPU windows while the environment executes the following actions (Figure 4).

Zero client state. We move the scheduling complexity of rollout overlap from the RL side to the reward server. The RL side keeps no reward-related state and streams each screenshot and action as soon as it is produced, in any order within one rollout or across rollouts, and the server reorders the stream and schedules the overlapped computation. The reward model thus attaches to an existing RL framework without changes to the RL-side code.

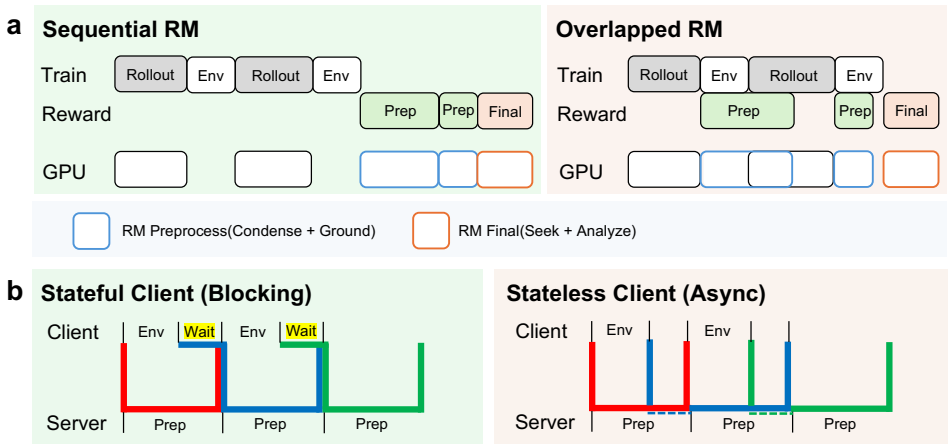


Figure 4: Reward server. **(a)** A sequential reward model judges only after the trajectory ends, while SeekJudge runs Condense and Ground during the rollout, leaving only the final stage on the critical path. **(b)** A stateful client blocks between steps to keep requests ordered, while a zero-state client streams out of order and the server reorders.

Table 1: CUAStepBench compared with existing reward benchmarks. *Traj. labels* gives the source of the trajectory verdict, and *Step labels* gives which steps carry a label and how many.

Benchmark	Traj. labels	Step labels	Labeled steps
AgentRewardBench (Lù et al., 2025)	Human	None	–
OmniGUIRewardBench (Li et al., 2026)	Rule	None	–
CUARewardBench (Lin et al., 2026)	Human	Key actions only	346
CUAVerifierBench (Rosset et al., 2026)	Human	Whole-trajectory process only	–
OS-Critic Bench (Wu et al., 2026)	None	Unexecuted candidate actions	738
CUAStepBench (ours)	Human	Every executed step	5,634

4 Experiments

4.1 CUAStepBench

To evaluate the step-level judgments of a reward model together with its trajectory verdicts, we build CUAStepBench, a human-annotated benchmark of 278 tasks over 177 applications. Every trajectory carries a human verdict, and beyond that verdict each step receives a human binary label of whether it is correct. Table 1 places CUAStepBench among existing reward benchmarks. Every task goal in CUAStepBench comes from a human-written task description.

We keep hard cases so that the benchmark separates judges. Two of the seven dimension scores, negative and positive evidence reliability, are the confidence that the trajectory failed and the confidence that it succeeded. We run SeekJudge with its teacher models over the candidate pool and keep the tasks with both confidences low. Low confidence on both sides marks a trajectory whose outcome the screenshots do not clearly determine. We then balance the retained tasks across trajectory lengths, applications and task types. Appendix J describes this curation from the candidate pool to the final 278 tasks. Only about 500 of the roughly 5K step labels are error steps, so we report F1 rather than accuracy at the step level, and F1 at the trajectory level as well for consistency.

4.2 Training and Evaluation Details

All RL environments come from the Chrome, Impress and OS domains of OSWorld (Xie et al., 2024), with 12 to 24 training and test tasks per domain. We train a separate policy for every domain with GRPO, and every reward is compared under identical training within one domain. Appendix K.2 lists the hyperparameters and the evaluation protocol. We report test success as the mean over the test evaluations taken every 15 steps during training and training success as the rule-

Table 2: Reinforcement-learning main results. Training (Train) and test (Test) success rates in %, both measured by rule-based evaluation under the protocol of Section 4.2.

		Chrome		Impress		OS		Avg	
Backbone	Reward	Train	Test	Train	Test	Train	Test	Train	Test
UI-TARS-1.5-7B	Rule-based	**41.33**	12.75	**43.00**	30.43	**55.32**	25.56	**46.55**	22.91
	CUAJudge	34.15	13.91	29.18	33.48	49.83	26.11	37.72	24.50
	OS-Themis	12.64	10.14	19.59	32.46	20.45	25.00	17.56	22.53
	SeekJudge	37.40	**16.23**	30.06	**38.84**	49.11	**28.89**	38.86	**27.99**
Qwen3-VL-8B	Rule-based	**54.89**	14.49	**43.84**	49.28	**69.73**	**26.67**	**56.15**	30.15
	CUAJudge	47.56	14.20	27.35	40.58	61.57	20.56	45.49	25.11
	OS-Themis	42.57	11.01	20.58	40.87	65.14	25.00	42.76	25.63
	SeekJudge	48.43	**20.58**	32.19	**50.14**	66.11	24.17	48.91	**31.63**

Table 3: Offline evaluation on CUAStepBench, trajectory- and step-level Precision, Recall and F1 in %, with step-level metrics over error steps. All judge rows are our runs, and – marks judges that give no step labels. † marks frameworks without native step-level judgments, whose step labels come from the step prompt of SeekJudge (Appendix M) on the same base model. DS + Gemini is DeepSeek-V4-Pro-0423 as the Seek agent and Gemini-3.0-Flash as the Analyze agent.

		Trajectory-level			Step-level		
Framework	Model	Prec	Recall	F1	Prec	Recall	F1
Human expert	–	80.3	84.1	82.2	–	–	–
Large models							
Codex	GPT-5.5	87.2	70.3	77.9	–	–	–
CUAJudge†	GPT-5-mini	78.6	60.7	68.5	49.5	30.6	37.9
SeekJudge	DS + Gemini	64.0	90.3	74.9	48.7	43.0	45.7
Small models							
OSReward	OS-Shepherd-9B	74.2	63.4	68.4	–	–	–
CUAJudge†	Qwen3-VL-8B	72.0	62.1	66.7	66.7	12.6	21.2
OS-Themis†	Qwen3-VL-8B	71.3	51.5	59.8	17.0	14.2	15.5
SeekJudge	Qwen3-VL-8B	56.6	96.4	71.3	41.4	20.4	27.3
SeekJudge	SeekJudge-9B	68.6	80.8	**74.2**	44.5	33.1	**38.0**

measured rate on the training tasks averaged over training steps. Across three pairs of repeated runs, the pooled standard deviation is about 2.0 points for test success and 2.2 for training success, so a reward gap averaged over two backbones and three domains has a standard error of 1.15 points.

4.3 Reinforcement Learning Main Results

Table 2 reports the test success rate of policies trained under each reward model. The results show that **(1)** Averaged over the six backbone-domain settings, SeekJudge exceeds native rule-based reward by 3.3 points of test success (about 2.9 times the standard error), while rule-based reward reaches the highest training success rate because the Train column is its own training signal. **(2)** Among model-based rewards, SeekJudge gives the highest average test success on both backbones. CUAJudge and OS-Themis enter RL through their native binary verdicts, and SeekJudge leads them by combining a finer-grained reward with a stronger judge. Appendix G measures reward hacking against the rule-based evaluation during training.

4.4 Offline Reward Benchmark Evaluation

We evaluate SeekJudge as a static judge on three reward benchmarks against CUAJudge, OS-Themis and the OS-Shepherd-9B judge from OSReward (Sun et al., 2026a), with the large-model CUAJudge on GPT-5-mini, the model of its release. CUAStepBench scores both the trajectory verdict

Table 4: Offline evaluation on AgentRewardBench and OmniGUIRewardBench, trajectory-level results in %. † marks results and model names quoted from their source papers. Rule reproduces the execution-rule labels of OmniGUIRewardBench by construction, so its entries are italic and excluded from the marking.

		AgentRewardBench			OmniGUIRewardBench		
Framework	Model	Prec	Recall	F1	Prec	Recall	F1
Rule†	–	83.8	55.9	67.1	*100.0*	*100.0*	*100.0*
WebJudge†	o4-mini	82.0	47.8	60.4	–	–	–
WebJudge†	Qwen + o4-mini	75.7	58.0	65.6	–	–	–
World-State-Model†	WSM-7B	71.2	72.2	71.7	–	–	–
CUAJudge	GPT-5-mini	75.2	53.6	62.6	90.5	78.0	83.8
OS-Themis	Qwen3-VL-8B	73.1	42.4	53.6	88.4	63.1	73.7
SeekJudge	Qwen3-VL-8B	46.3	90.1	61.2	67.3	90.8	77.3
SeekJudge	SeekJudge-9B	74.2	79.2	**76.6**	85.3	85.0	**85.1**

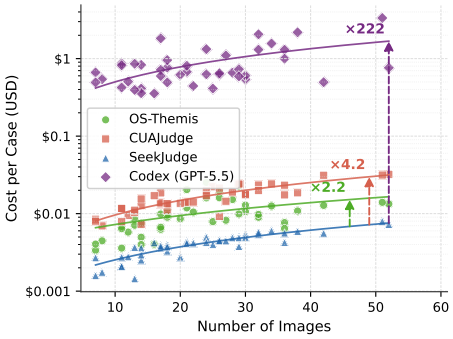


Figure 5: Per-case judging cost against the number of images, one marker per case with a linear fit per judge on a log cost axis.

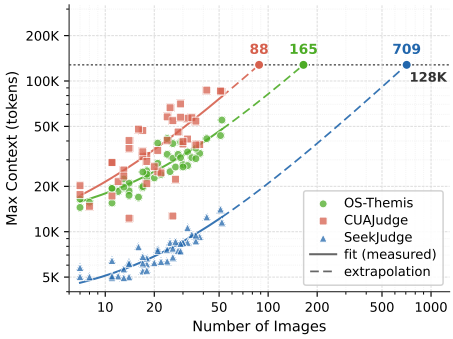


Figure 6: Peak context size against the number of images, continued dashed to the 128K window.

and the step labels, while AgentRewardBench and OmniGUIRewardBench score the trajectory verdict alone. The results show that **(1)** SeekJudge-9B is competitive with every baseline except the agentic Codex run. It exceeds the Rule baseline on AgentRewardBench by 9.5 F1, reaches a higher trajectory F1 than the GPT-5-mini-based CUAJudge on all three benchmarks, and matches its step-level F1 on CUAStepBench, 38.0 against 37.9. **(2)** With the untrained Qwen3-VL-8B backbone, the SeekJudge framework already exceeds OS-Themis on the same backbone in trajectory F1 on all three benchmarks, with high recall at the cost of precision. All SeekJudge entries use one shared threshold, and their precision and recall depend on it. Appendix A plots the threshold-free ROC curve of SeekJudge-9B against every baseline's operating point.

4.5 Efficiency: Cost, Context and Training Overhead

Cost. We compute the cost of every judge on the same 43 cases under one cost model that converts open-source and closed-source calls to a common dollar cost per token, with the token prices and the cache rule given in Appendix H. Figure 5 shows that SeekJudge judges at the lowest cost at every image count, and its cost gap to every other judge widens as the number of images grows. CUAJudge incurs extra GPT-5-mini calls, and the agentic Codex baseline costs two orders of magnitude more.

Context. The peak context of a judge sets the longest trajectory that fits in its context window. Figure 6 plots, for each case, the token count of the largest single request that a judge issues against the number of images. At 52 images the fitted lines give about 12K tokens for SeekJudge, 48K for OS-Themis and 80K for CUAJudge, about 4× and 6× that of SeekJudge. Extrapolated, the lines reach 128K tokens at about 90 images for CUAJudge, 160 for OS-Themis and 700 for SeekJudge.

Table 5: Rollout time per interaction round in seconds, Qwen3-VL-8B trained with the SeekJudge reward, judging after each trajectory ends (Sequential) or with rollout overlap. Overhead is the part of Total outside the actor's and environment's own execution. Overall averages the domains.

	Chrome		Impress		OS		Overall	
Scheduling	Total	Overhead	Total	Overhead	Total	Overhead	Total	Overhead
Sequential	61.0	24.1	55.3	20.6	59.2	10.2	58.5	18.3
Rollout overlap	**46.6**	**8.1**	**48.0**	**9.4**	**51.8**	**6.2**	**48.8**	**7.9**
Reduction	*−23.6%*	*−66.4%*	*−13.2%*	*−54.4%*	*−12.5%*	*−39.2%*	*−16.6%*	*−56.8%*

Table 6: Reward-granularity ablation, training and test success rates in % of UI-TARS-1.5-7B on the Impress domain.

Reward	Granularity	Train	Test
Rule-based	Boolean	**43.00**	30.43
SeekJudge	Boolean	21.08	35.94
SeekJudge	Continuous	24.25	33.62
SeekJudge	Continuous + step-level	30.06	**38.84**

Table 7: Framework ablation on a pair of strong models, DeepSeek-V4-Pro-0423 as the Seek agent and Gemini-3.0-Flash as the Analyze agent, evaluated on CUAStepBench.

Framework component			F1 (%)	
Condense	Ground	Seek–Analyze	Traj	Step
✓	✓	✓	**74.9**	**45.7**
✓	✓	×	72.6	38.7
×	✓	×	67.1	34.3
✓	×	×	72.3	40.1

Training overhead. We train Qwen3-VL-8B with the SeekJudge reward in each domain twice, judging only after each trajectory ends or with the rollout overlap of Section 3.3, and measure the round time and the overhead outside the actor's and environment's own execution. Most of this overhead is the time that training waits for the reward model to return its judgments. Table 5 shows that rollout overlap reduces the overhead by 56.8% overall, from 18.3 s to 7.9 s per round.

4.6 Ablations

Reward granularity. Table 6 trains UI-TARS-1.5-7B on Impress with a Boolean, a continuous, and a continuous plus step-level SeekJudge reward. Adding the step term of Eq. 1 to the continuous score gives the default reward and raises test success from 33.62 to 38.84, the highest in the table.

Framework ablation. Table 7 removes components of SeekJudge at inference and evaluates each variant on CUAStepBench with a pair of strong models. We use this pair rather than SeekJudge-9B to avoid a distribution shift caused by evaluating SeekJudge-9B differently from how it was trained. Condense and the Seek–Analyze loop both contribute. Without the loop, removing Condense lowers trajectory F1 by 5.5 points and step F1 by 4.4 points. Adding the loop to Condense and Ground raises trajectory F1 by 2.3 points and step F1 by 7.0 points.

5 Conclusion

We revisited long-trajectory judging for computer-use agents as the decomposition into localization and extraction, showing that, even with the decisive screenshot always present, adding more screenshots of the same trajectory to one forward pass degrades the reading of the decisive evidence. SeekJudge realizes this decomposition with four agents on a single 9B backbone. To our knowledge, it is the first model-based reward to match or surpass native rule-based reward on downstream test success in online RL, at a fraction of closed-source cost with a 4–$6\times$ smaller peak context. SeekJudge, CUAStepBench and the rollout-overlapped reward server together make model-based reward a practical substitute for rule-based reward and open a path to reinforcement learning in environments that rules cannot instrument.

6 The Use of Large Language Models

We use LLMs to search related works and identify relevant literature connections across the field of computer-use agents and reward modeling for reinforcement learning, and to aid and polish writing, including spell checking, finding appropriate vocabulary to express intended meanings.

7 Reproducibility Statement

Section 3.2 specifies the teacher models and the two prompt searches that build the distillation data, and Section 4.1 describes how the tasks of CUAStepBench are selected and labeled. The prompts that drive exploration and scoring, including the criteria prompt that produces the dense labels, are described in Appendix M. Appendix K.2 lists the RL hyperparameters and the evaluation protocol, and Appendix K.3 gives the rule that combines the per-step scores into the trajectory reward. Appendix H gives the token prices, the closed-source rates, and the prefix-cache rule that we apply uniformly to every method when computing inference cost. Appendix E describes the leakage control for the distillation data, the prompt search, and the score regression. We will release the training data, the SeekJudge-9B model, CUAStepBench, and the code of SeekJudge and its reward server.

References

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.11896.

Hyungjoo Chae, Seonghwan 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 Han, Taeyoon Kwon, Minju Kim, Beongwoo Kwak, Dongjin Kang, and Jinyoung Yeo. Web-Shepherd: Advancing PRMs for reinforcing web agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2505.15277.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. Agentic reward modeling: Verifying GUI agent via progressive trajectory-grounded interaction. *arXiv preprint arXiv:2602.00575*, 2026.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuqing Yang, Yuanchun Li, Rui Tan, Mo Li, and Lili Qiu. ProRe: A proactive reward system for GUI agents via reasoner–actor collaboration. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.21823.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 28091–28114, 2023. arXiv:2306.06070.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pp. 11642–11662, 2024. arXiv:2403.07718.

Jihao Gu, Qihang Ai, Yingyao Wang, Pi Bu, Jingxuan Xing, Yue Cao, Zekun Zhu, Wei Jiang, Ziming Wang, Yingxiu Zhao, Ming-Liang Zhang, Jun Song, Yuning Jiang, and Bo Zheng. Mobile-R1: Towards interactive capability for VLM-based mobile agent via systematic training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 30802–30820, 2026. arXiv:2506.20332.

Haojie Hao, Longkun Hao, Yihang Lou, Yan Bai, Zhenyang Li, Zhichao Yang, Dongshuo Huang, Hongyu Lin, Lanqing Hong, Jiakai Wang, and Xianglong Liu. StainFlow: Entity-stain tracking and evidence linking for process rewards in GUI agents. *arXiv preprint arXiv:2606.07027*, 2026.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6864–6890, 2024. arXiv:2401.13919.

Yifei He, Rui Yang, Hao Bai, Tong Zhang, and Han Zhao. PRO-CUA: Process-reward optimization for computer use agents. *arXiv preprint arXiv:2605.29119*, 2026.

Yoonna Jang, Kisu Yang, and Isabelle Augenstein. Expanding computation spaces of LLMs at inference time. *arXiv preprint arXiv:2509.24884*, 2025.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 881–905, 2024. arXiv:2401.13649.

Wei Li, William Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on UI control agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.03679.

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. *arXiv preprint arXiv:2603.19191*, 2026.

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

Haojia Lin, Xiaoyu Tan, Yulei Qin, Zihan Xu, Yuchen Shi, Zongyi Li, Gang Li, Shaofei Cai, Siqi Cai, Yuzheng Cai, Chaoyou Fu, Ke Li, and Xing Sun. CUARewardBench: A benchmark for evaluating reward models for computer-using agents. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2510.18596.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. GUIOdyssey: A comprehensive dataset for cross-app GUI navigation on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22404–22414, 2025. arXiv:2406.08451.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. In *Second Conference on Language Modeling (COLM)*, 2025. arXiv:2504.08942.

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 (COLM)*, 2024. arXiv:2404.06474.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2411.02337.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2405.14573.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. The art of building verifiers for computer use agents. *arXiv preprint arXiv:2604.06240*, 2026.

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

Marta Sumyk and Oleksandr Kosovan. Reinforcement learning for computer-use agents with autonomous evaluation. *arXiv preprint arXiv:2606.24515*, 2026.

Qiushi Sun, Kanzhi Cheng, Yian Wang, Bowen Yang, Hang Yan, Liheng Chen, Fangzhi Xu, Zichen Ding, Nuo Chen, Jialin Cao, Xingdong Gong, Zehao Li, Kaiming Jin, Xinfeng Yuan, Zhoumianze Liu, Jingyang Gong, Zhangyue Yin, Jiahui Gao, Zhiyong Wu, Tianbao Xie, Jianbing Zhang, Ben Kao, and Lingpeng Kong. OSReward: Instituting standardized evaluation for cross-platform computer-use reward models. *arXiv preprint arXiv:2607.28609*, 2026a.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. SEAgent: Self-evolving computer use agent with autonomous learning from experience. In *International Conference on Machine Learning (ICML)*, 2026b. arXiv:2508.04700.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. CUA-Gym: Scaling verifiable training environments and tasks for computer-use agents. *arXiv preprint arXiv:2605.25624*, 2026.

Hengyi Wang, Haizhou Shi, Shiwei Tan, Weiyi Qin, Wenyuan Wang, Tunyu Zhang, Akshay Nambi, Tanuja Ganu, and Hao Wang. Multimodal needle in a haystack: Benchmarking long-context capability of multimodal large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pp. 3221–3241, 2025a. arXiv:2406.11230.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 9426–9439, 2024. arXiv:2312.08935.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, et al. OpenCUA: Open foundations for computer-use agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b. arXiv:2508.09123.

Zhenyu Wu, Jingjing Xie, Zehao Li, Bowen Yang, Qiushi Sun, Zhaoyang Liu, Zhoumianze Liu, Yu Qiao, Xiangyu Yue, Zun Wang, and Zichen Ding. OS-Oracle: A comprehensive framework for cross-platform GUI critic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27514–27524, 2026. arXiv:2512.16295.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2404.07972.

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 Lab blog, July 2025. URL `https://xlang.ai/blog/osworld-verified`.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? Assessing the current state of web agents. In *Second Conference on Language Modeling (COLM)*, 2025. arXiv:2504.01382.

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. *arXiv preprint arXiv:2602.10356*, 2026.

Chenyu Yang, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, Wenhai Wang, Yu Qiao, Xizhou Zhu, and Jifeng Dai. ZeroGUI: Automating online GUI learning at zero human cost. *arXiv preprint arXiv:2505.23762*, 2025.

Rui Yang, Qianhui Wu, Yuxi Chen, Hao Bai, Wenlin Yao, Hao Cheng, Baolin Peng, Huan Zhang, Tong Zhang, and Jianfeng Gao. OpenWebRL: Demystifying online multi-turn reinforcement learning for visual web agents. *arXiv preprint arXiv:2606.02031*, 2026.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan. Mobile-Agent-v3: Fundamental agents for GUI automation. *arXiv preprint arXiv:2508.15144*, 2025.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8938–8968, 2024. arXiv:2407.15711.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, et al. OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks. *arXiv preprint arXiv:2606.29537*, 2026.

Congmin Zheng, Xiaoyun Mo, Xinbei Ma, Qiqiang Lin, Yin Zhao, Jiachen Zhu, Xingyu Lou, Jun Wang, Zhaoxiang Wang, Weiwen Liu, Zhuosheng Zhang, Yong Yu, and Weinan Zhang. Adaptive milestone reward for GUI agents. *arXiv preprint arXiv:2602.11524*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 46595–46623, 2023. arXiv:2306.05685.

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 *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.13854.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, et al. Establishing best practices in building rigorous agentic benchmarks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2507.02825.

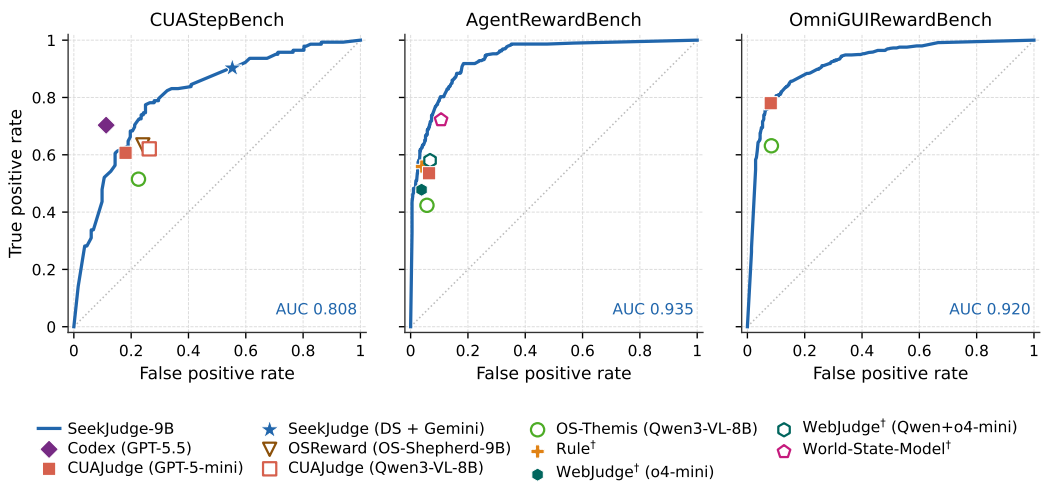


Figure 7: Trajectory-level ROC curves of SeekJudge-9B on the three offline benchmarks, from a threshold sweep over the regressor score. Every other judge returns a binary verdict and appears as one point. Filled markers are closed-source or large models and hollow markers small open models. † marks results quoted from source papers.

A Trajectory-Level ROC Curves

Tables 3 and 4 compare judges at one operating point each, and the threshold on the regressor score determines where on its curve the point of SeekJudge lies. Figure 7 removes the threshold. It sweeps the threshold on the regressor score of SeekJudge-9B under the joint fit of Appendix E.2 and plots the resulting ROC curve on each benchmark, with every other judge as the single point that its binary verdict defines. The false positive rate of a point follows from its precision, recall and accuracy, and for the results quoted from source papers on AgentRewardBench, which report no accuracy, from the positive rate of the test split. The area under the curve is 0.808 on CUAStepBench, 0.935 on AgentRewardBench and 0.920 on OmniGUIRewardBench. The CUAStepBench entries of Table 3 average three repeats while the sweep uses one run, so the operating point of that table lies near rather than on the curve.

Three readings follow. First, on AgentRewardBench every baseline lies below the curve, so at the false positive rate of each baseline some threshold of SeekJudge-9B reaches a higher recall, 70 against 54 at the false positive rate of CUAJudge and 80 against 72 at that of World-State-Model. Second, on OmniGUIRewardBench the CUAJudge point lies on the curve and the OS-Themis point below it. Third, on CUAStepBench the agentic Codex run lies above the curve and CUAJudge with GPT-5-mini close to it, while OSReward, CUAJudge with Qwen3-VL-8B and OS-Themis lie below.

B Per-Step Reward Variant

SeekJudge-step keeps the step term instead of collapsing it. Every step shares the trajectory reward R_i of Eq. 1 and adds its own step score, with the same weight $w_{\text{step}} = 2$, and the GRPO baseline pools every step of the $G = 8$ rollouts in a group,

$$R_{i,k} = R_i + w_{\text{step}}\, S^{\text{step}}_{i,k}, \tag{2}$$

$$\hat{A}_{i,k} = \frac{R_{i,k} - \text{mean}(\mathcal{R})}{\text{std}(\mathcal{R})}, \qquad \mathcal{R} = \{R_{j,l} : 1 \le j \le G,\ 1 \le l \le N_j\}. \tag{3}$$

Unlike Eq. 4, the right side of Eq. 3 depends on k, and the numerator and the baseline share the per-step scale.

Table 8 compares the variant with SeekJudge under the protocol of Table 2. The step term changes test success by -1.7, $+1.5$ and -10.0 points on UI-TARS Chrome, Impress and OS and by -4.9,

Table 8: Per-step reward variant (SeekJudge-step) against the trajectory reward (SeekJudge). Training (Train) and test (Test) success rates in % under the RL protocol of Section 4.2.

		Chrome		Impress		OS		Avg	
Backbone	Reward	Train	Test	Train	Test	Train	Test	Train	Test
UI-TARS-1.5-7B	SeekJudge	37.40	16.23	30.06	38.84	49.11	28.89	38.86	27.99
	SeekJudge-step	33.23	14.49	29.30	40.29	47.95	18.89	36.83	24.56
Qwen3-VL-8B	SeekJudge	48.43	20.58	32.19	50.14	66.11	24.17	48.91	31.63
	SeekJudge-step	46.07	15.65	32.63	45.51	68.08	34.44	48.93	31.87

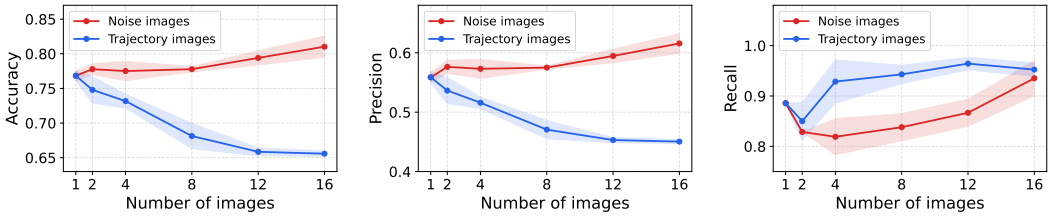


Figure 8: Accuracy, precision and recall against the number of images in one forward pass, with the decisive screenshot always present. Padding uses other screenshots of the same trajectory (*blue*) or noise images of equal token budget (*red*).

-4.6 and $+10.3$ points on Qwen Chrome, Impress and OS, which lowers the UI-TARS average by 3.4 points and leaves the Qwen average within 0.3 point of SeekJudge. Adding the step term this way therefore gives no consistent gain. We leave per-step credit assignment to future work.

C Multi-Image Interference

We test whether reading a detail from an image degrades when many images share one forward pass. To isolate this extraction from localization, we keep the 123 of 278 CUAStepBench cases whose completion can be determined from a single screenshot, selected with the agentic Codex judge on GPT-5.5, and always place that decisive screenshot in the input. The judge under test is CUAJudge on the Qwen3-VL-8B backbone, which reads all images of a case in one forward pass. Holding the decisive screenshot fixed, we pad the input with other screenshots of the same trajectory in order (blue) or with noise images of the same token budget and no readable content (red). Figure 1(b) reports F1, and Figure 8 reports accuracy, precision and recall.

The results show that **(1)** for this judge, competing content rather than context length drives the drop. Blue and red carry the same context length, so their gap isolates trajectory content from length. Blue F1 falls from 68 to 61 as images accumulate, even though the decisive image is always present, while red stays level up to 8 images and then rises, so the added trajectory detail degrades the verdict and length alone does not. **(2)** More images bias the judge toward accepting the trajectory. Splitting blue F1 into its terms, recall rises slightly while precision drops from 56 to 45.

The setting matches multimodal needle-in-a-haystack retrieval, which also degrades as images accumulate (Wang et al., 2025a). One possible explanation for the rise of the noise-image curve is the effect reported by Jang et al. (2025), where extra uninformative tokens widen the model's computation and add parallel scratch space.

D Adaptive Compute

SeekJudge spends more compute on longer trajectories. Figure 9 groups trajectories into five equal-size bins by step count and reports the mean number of tool calls that the Seek agent issues in each. Tool calls grow slowly with trajectory length. One round may issue several questions, so the longest bin averages more calls than the round cap of 8.

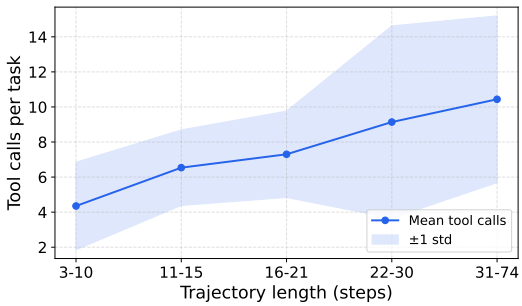


Figure 9: Mean number of tool calls per case against trajectory length, with a ± 1 standard deviation band around the mean.

Table 9: Leakage probe for the score regression, reporting trajectory-level F1 (%) of the trained SeekJudge-9B backbone. Each row is a regression fit on a different set of benchmarks and then read on all three. Row 1 is the joint fit on all three benchmarks that we use throughout the paper (light yellow , small overfitting risk). Row 2 is a five-fold cross-fit over the same three benchmarks, with every trajectory scored by a fold model that never trained on it and the threshold set on that fold's own training rows. Rows 3–5 each fit on a single benchmark. A diagonal cell fits and evaluates on the same benchmark (light red , highest overfitting risk), while an off-diagonal cell evaluates on a benchmark absent from its fit (light green , lowest overfitting risk).

	Evaluated on (trajectory F1)		
Regression fit on	AgentRewardBench	CUAStepBench	OmniGUIRewardBench
All three (ours)	76.6	74.2	85.1
Five-fold cross-fit	75.9	75.8	84.6
AgentRewardBench	76.7	70.1	83.8
CUAStepBench	76.8	74.6	85.0
OmniGUIRewardBench	75.7	74.1	85.3

E Leakage Control

Information from an evaluation target can leak back into SeekJudge at three points, the distillation training data, the prompt search, and the score regression. We control each in turn.

E.1 Distillation Data and Prompt Search

Training data. The distillation set that trains SeekJudge shares no task goal with any offline benchmark or online RL environment, so the model is never trained on a case that it is later evaluated on.

Prompt search. A prompt search can overfit the set that it is scored against. We therefore score both searches on fixed anchor sets and never on benchmark ground truth. The criteria prompt is searched against about 20 human-annotated trajectories that anchor what counts as success. The seek prompt is searched against the progress descriptions that Codex with GPT-5.5 writes for about 1K trajectories, and these descriptions guide what to ask and when to stop. Claude Code runs both searches, reading the failures of each case and revising the prompt. All remaining supervision comes from teacher-model distillation.

E.2 Score Regression

Score regression. The regressor that maps the seven dimension scores to a trajectory score is an XGBoost model of 200 trees of depth 2, with a learning rate of 0.03, an L2 penalty of 200 and a

minimum child weight of 10. Each feature is a dimension score divided by 100, with the sign flipped for the dimensions that lower success, and every feature carries a monotone constraint, so the score cannot decrease when any dimension improves. The model reads only these seven scalar features. One model and one threshold serve every benchmark. The threshold is the value that maximizes Youden's J on the fitted rows pooled across benchmarks and equals 0.374 on the probability scale, so SeekJudge never selects an operating point per benchmark, and each benchmark receives the same total sample weight in the fit, so a large benchmark does not dominate a small one.

Two fits. Unless noted otherwise the whole paper uses only two fits of this regressor. The first serves offline evaluation, and we apply it to every offline benchmark, to the trained SeekJudge-9B and the untrained Qwen3-VL-8B backbone alike. The second serves RL. GRPO compares rollouts inside a group that shares one goal, so a dimension that the goal fixes on its own, such as goal infeasibility, takes the same value across the whole group and cannot separate a stronger trajectory from a weaker one. The second fit therefore drops the two goal-level dimensions and reads only the other five dimension scores, with no task identity. We call it the RL calibrator. It is fit once, before training, with the hyperparameters above, on about 1K rollouts in total of the untrained base policy over the tasks of the three domains, labeled by the environment's rule-based evaluation, and is then frozen. With five features and depth-2 trees against about 1K samples, it has little capacity to store a task-specific solution. GRPO also normalizes the reward within a group, so an offset in S^{traj} cancels, a change of its scale only reweights the trajectory term against the step term, and the policy sees the relative order of the rollouts of one goal rather than their absolute scores. Appendix F replaces this calibrator with the offline fit, which sees no rollout of the RL policy.

Adaptation. The seven dimension scores are fixed by the prompts, and the mapping from these scores to a verdict is the only part of the pipeline that differs between the two fits. On UI-TARS-1.5-7B Impress, RL with the offline fit in place of the RL calibrator still surpasses rule-based reward (Appendix F).

To probe the residual risk we vary how the regressor is fit and read every fit on all three benchmarks, in Table 9. The setting reported throughout the paper fits jointly on all three, so every reported number reads a benchmark that its fit has seen. Against it we place two kinds of held-out fit. The first is a five-fold cross-fit over the same three benchmarks. Every trajectory is scored by a fold model that never trained on it, and the threshold of each fold comes from that fold's own training rows. The second is three single-benchmark fits. Each produces a diagonal cell that fits and evaluates on the same benchmark and off-diagonal cells that evaluate on benchmarks absent from the fit. In a diagonal cell the regressor has seen the target distribution, so this cell is the highest-risk reading. The cross-fit removes that exposure while keeping the fitting distribution, and an off-diagonal cell removes it while also shifting the fitting distribution to other benchmarks and shrinking the fitting set.

Two readings of Table 9 bound the risk. First, the diagonal fits carry the highest risk, yet none of them leads its column by a meaningful margin. On AgentRewardBench the clean transfer fit on CUAStepBench even slightly exceeds it, and on the other two benchmarks it exceeds the nearest off-diagonal transfer by at most half a point. Second, the cross-fit, which never scores a trajectory with a model that trained on it, reaches 75.9, 75.8 and 84.6, within 0.7 points of the joint fit on AgentRewardBench and OmniGUIRewardBench and 1.6 points above it on CUAStepBench, so held-out scoring lowers the reported numbers by under one point at most. The single-benchmark transfers add cross-benchmark shift and a smaller fitting set on top of held-out scoring, and even under every fit that excludes the target benchmark, SeekJudge-9B still reaches 75.7 on AgentRewardBench, 70.1 on CUAStepBench and 83.8 on OmniGUIRewardBench, above every baseline on the first two except the agentic Codex run on CUAStepBench (71.7 for World-State-Model and 68.5 for CUAJudge) and equal to CUAJudge on the third (83.8, Tables 3 and 4).

F RL Reward with the Offline Fit

The RL calibrator of Appendix E.2 is fit on rollouts from the RL environments. To isolate the judge from this fit, we train UI-TARS-1.5-7B on Impress with the offline fit in place of the RL calibrator and keep everything else of the continuous score with the step term unchanged. The offline fit sees no rollout of the RL policy.

Table 10: Rule-based reward and SeekJudge reward (continuous score with the step term) under the two fits of the score regression on UI-TARS-1.5-7B Impress, success rates in %. The Rule-based and RL calibrator rows repeat the corresponding rows of Table 6.

Reward	Fit	Train	Test
Rule-based	–	**43.00**	30.43
SeekJudge	RL calibrator	30.06	**38.84**
SeekJudge	Offline fit	30.17	36.52

Table 11: Change of the pass rates over the 75 RL training steps on the Qwen3VL-8B backbone, in points and averaged over Chrome, Impress, and OS. J is the pass rate under the reward model and G is the pass rate under the rule-based evaluation, both on the same training rollouts. $\Delta J - \Delta G$ is the part of the rise in J that the rule-based evaluation does not confirm.

Reward	ΔJ	ΔG	$\boldsymbol{\Delta J - \Delta G}$	$J - G$ at start	$J - G$ at end
Rule-based	$+16.0$	$+16.0$	$\mathbf{0.0}$	0.0	0.0
SeekJudge	$+13.7$	$+9.5$	$\mathbf{+4.2}$	$+8.2$	$+12.4$
CUAJudge	$+30.1$	$+5.1$	$\mathbf{+25.0}$	-7.4	$+17.7$
OS-Themis	$+7.9$	$+5.0$	$\mathbf{+2.9}$	-16.9	-13.9

Table 10 gives the result. With the offline fit, SeekJudge reaches 36.52 test success, 6.1 points above rule-based reward, and the RL calibrator raises it further to 38.84. In this setting, SeekJudge thus surpasses rule-based reward without the RL calibrator.

G Reward Hacking during RL

During each RL run, every training rollout receives two verdicts, one from the reward model that the policy optimizes and one from the rule-based evaluation of OSWorld, which is also the evaluation used at test time. We write J for the pass rate under the reward model and G for the pass rate under the rule-based evaluation, both smoothed by an exponential moving average with weight 0.95. When the policy learns behavior that the reward model accepts and the rule-based evaluation rejects, J rises faster than G. Figure 10 plots both pass rates for each reward model and application on the Qwen3VL-8B backbone, and Table 11 reports their changes over the 75 training steps. Each panel uses the same runs as Table 2. For CUAJudge we recompute J from the verdicts and rewards in the training log.

The column $\Delta J - \Delta G$ of Table 11 is the part of the rise in J that the rule-based evaluation does not confirm. Averaged over the three applications, it is 4.2 points for SeekJudge, 25.0 points for CUA-Judge, and 2.9 points for OS-Themis. The rule-based reward optimizes the rule-based evaluation directly, so its value is 0. SeekJudge raises J by 13.7 points and G by 9.5 points, while CUAJudge raises J by 30.1 points and G by 5.1 points. OS-Themis stays below the rule-based evaluation throughout, with a gap $J - G$ of -16.9 points at the start and -13.9 points at the end, and raises J by 7.9 points. Part of the gain under SeekJudge is not confirmed by the rule-based evaluation, and this part is smaller than under CUAJudge.

H Inference Cost Model

This section gives the full cost model behind the cost comparison of Section 4.5. Every judge's per-case cost is the sum of an open-source token bill and a closed-source API bill, measured on the 43 cases shared by all four judges.

Token billing. We bill the self-deployed models by tokens rather than by measured GPU time. GPU time depends on how well each framework parallelizes its requests, so it mixes the concurrency engineering of a serial framework such as OS-Themis into that framework's reported cost.

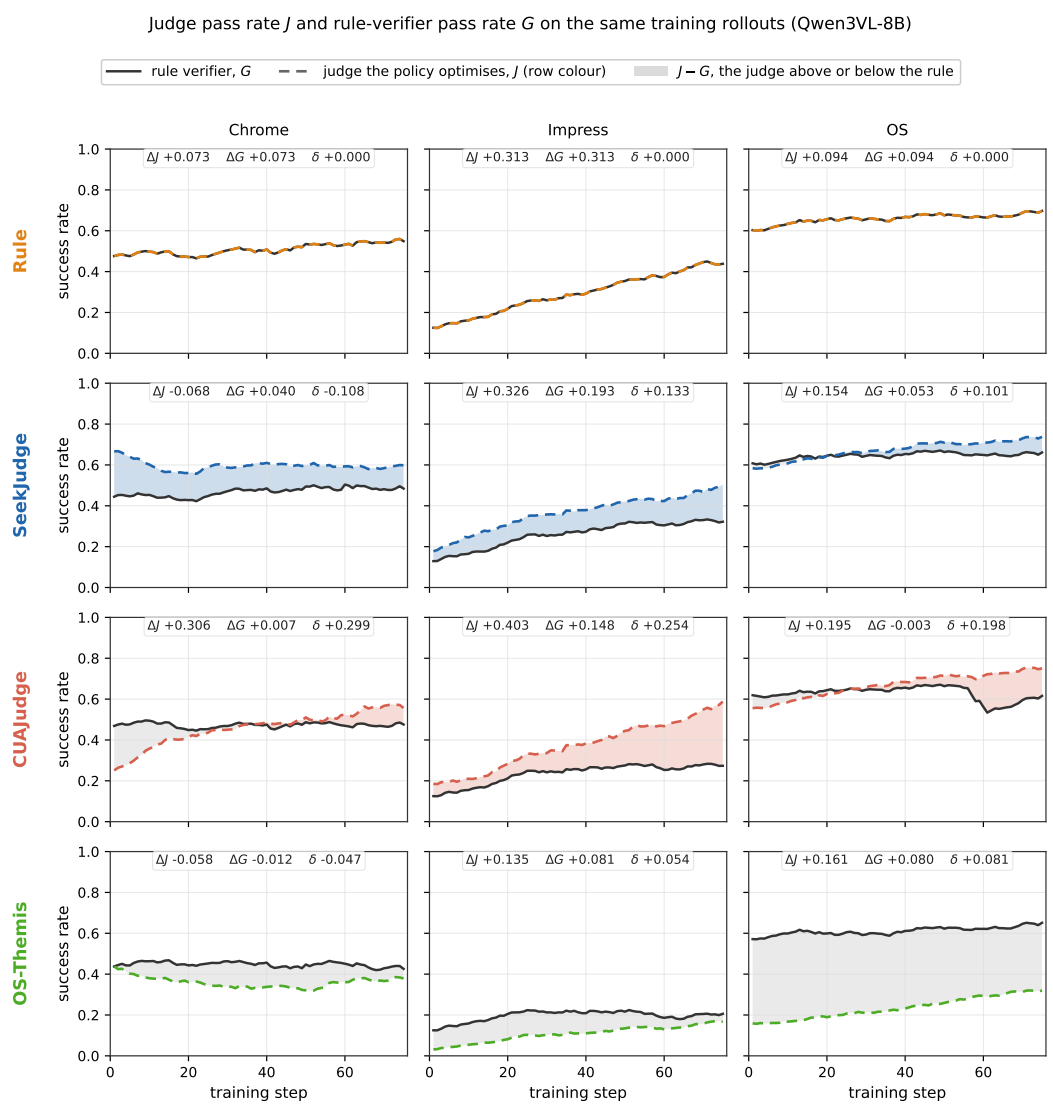


Figure 10: Pass rates on the same training rollouts during RL on the Qwen3VL-8B backbone. The black line is the rule-based evaluation G, the dashed line is the reward model J, and the shading is the gap between them.

Token prices. Open-source agents are billed at the public serving rate of a Qwen3.5-9B or Qwen3-VL-8B class model. This rate is \$0.04 per million input tokens and \$0.20 per million output tokens.[1] These prices cover the agents of SeekJudge and the Selector of OS-Themis. They also cover the key-image identification of CUAJudge, because we run it on an open-source model as in its official low-cost setting. Closed-source calls are billed at their provider's published price. The GPT-5-mini calls of CUAJudge are charged at the official OpenAI rate of \$0.25 per million input tokens and \$2.00 per million output tokens, and the agentic Codex baseline is charged at the GPT-5.5 API spend recorded for each run.

Prefix caching. We apply one uniform rule to every judge. A self-deployed small model serves a narrow, repetitive workload, so its requests within one case share a long common prefix that the KV cache returns at almost no cost. We therefore treat the prefix that a request shares with an earlier request in the same case as a cache hit, and bill unique input and all output in full. For the Seek

[1] https://deepinfra.com/blog/qwen3-5-9b-api-benchmarks

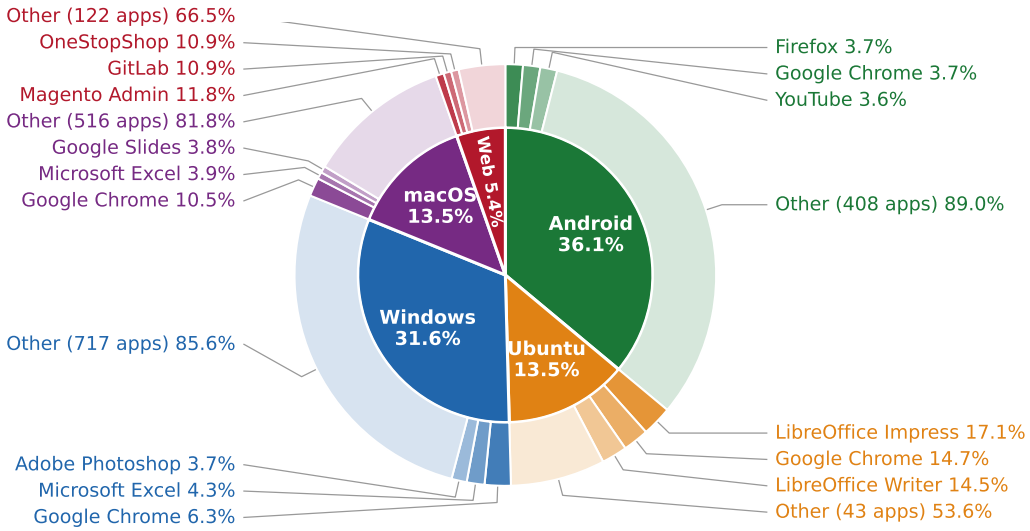


Figure 11: Environments and applications of the distillation trajectories. The inner pie splits the trajectories by environment, and the outer ring splits each environment into its three most frequent applications and Other. A trajectory that uses k applications counts $1/k$ toward each. Each label gives the share of an application within its environment.

agent in SeekJudge and the Selector in OS-Themis, whose input is dominated by one long, reused context, this counts input tokens only for the single longest request of the case and treats the input of every other request as a cache hit at zero cost. CUAJudge issues a single turn per case and has no repeated prefix to reuse, so the same rule gives it no discount. The cache covers only input encoding, so output tokens are always charged in full.

Validation on deployment logs. We verify this accounting on isolated per-agent logs from our sglang deployment. Within one case, each turn reports a cached-token count equal to the full token count of the preceding request, so the entire prior context is served from the prefix cache and only the marginal new tokens are re-encoded. Summed over the case, the freshly encoded input therefore equals the single longest request, and the accounting above charges exactly this request.

I Composition of the Distillation Data

Table 12 breaks the 10,905 distillation training tasks down by environment. The Android trajectories come from GUI-Odyssey (Lu et al., 2025), Mobile-R1 (Gu et al., 2026) and AndroidControl (Li et al., 2024), the Windows, macOS and Ubuntu trajectories from AgentNet (Wang et al., 2025b), and the web trajectories from WebArena (Zhou et al., 2024), VisualWebArena (Koh et al., 2024), WorkArena (Drouin et al., 2024), AssistantBench (Yoran et al., 2024) and Online-Mind2Web (Xue et al., 2025). The tasks cover 1,473 distinct applications after merging different spellings of one application, with a long tail. Google Chrome, the most frequent named application, appears in 1,569 tasks, and on Android the most frequent application accounts for under 4% of that environment. Figure 11 shows the environment split together with the three most frequent applications of each environment.

J Human Curation of CUAStepBench

Human curation starts from over 300 candidate tasks. The same annotator checks every label twice. A Codex run then audits the labels and flags 51 of them as suspicious, and two annotators review each flagged case. They correct 12 labels, 4.3% of the final benchmark, and discard the 7 cases that remain disputed after discussion. The same Codex configuration appears as a judge in Table 3.

Table 12: Composition of the distillation training set by environment. Proportion is over all trajectories, and Apps counts distinct applications, or distinct sites for Web.

Environment	Trajectories	Proportion (%)	Distinct tasks	Apps
Android	4,000	36.1	4,000	411
Windows	3,503	31.6	3,503	720
Ubuntu	1,500	13.5	1,500	46
macOS	1,497	13.5	1,497	519
Web	594	5.4	405	125
Total	11,094	100.0	10,905	1,473

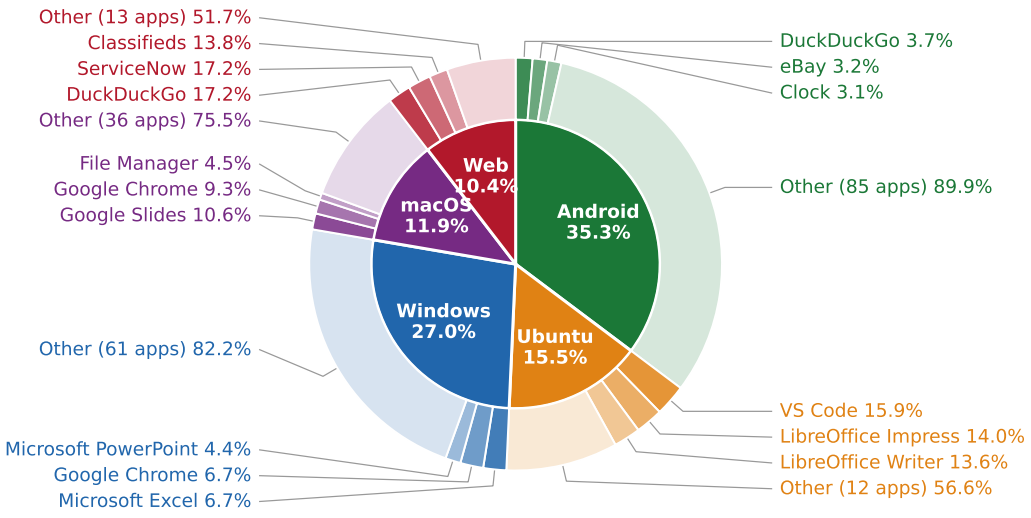


Figure 12: Environments and applications of the 278 CUAStepBench trajectories. The inner pie splits the trajectories by environment, and the outer ring splits each environment into its three most frequent applications and Other.

We further prune cases whose error cause repeats one already covered and balance the remaining trajectories across error types. The cost of running multi-round judges under every configuration that we evaluate sets the final size at 278 tasks. An independent expert who took no part in labeling or curation judges all 278 trajectories blind, from the same instruction, screenshots and actions that a judge receives. Against the final labels, the expert reaches 80.9% accuracy, 82.2 trajectory F1 , listed as Human expert in Table 3. Each final label, unlike this single blind pass, went through the double check and, when flagged, the two-annotator review described above.

Two properties of this construction matter for the comparison. The uncertainty filter keeps the cases that leave the teacher pipeline unsure and discards the cases that it judges with confidence, whether rightly or wrongly, and every judge in Table 3 is scored on the same retained set. The Codex audit only flags labels for review, and a label changes only after two annotators confirm the change. AgentRewardBench, which is human-labeled and whose cases we did not select, gives a second reading. There SeekJudge-9B leads CUAJudge by 14.0 points, against 5.7 points on CUAStepBench (Tables 3 and 4).

Figure 12 breaks the 278 CUAStepBench trajectories down by environment and application. Android contributes 98 trajectories, Windows 75, Ubuntu 43, macOS 33 and Web 29, over 177 distinct applications. These proportions are close to those of the distillation data in Figure 11, except that the Web share of 10.4% is about twice the 5.4% of the distillation data.

The disagreement study of Figure 1(a) uses the same kind of review. A Codex judge runs over every trajectory of OmniGUIRewardBench, and a human re-checks each of its 71 disagreements with the

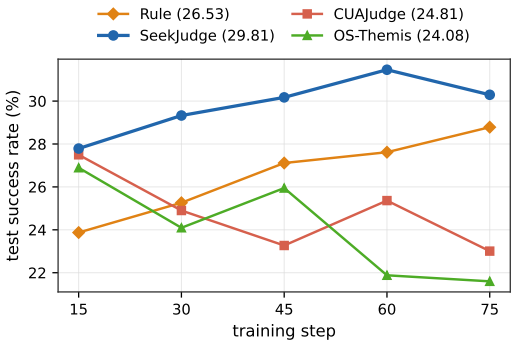


Figure 13: Test success during RL at training steps 15, 30, 45, 60 and 75, averaged over the six backbone-domain settings of Table 2. The value in the legend is the mean of each curve over these five steps, which equals the average test success of that reward in Table 2.

rule without knowing which verdict comes from the rule and which from the judge, agreeing with the judge on 48.

K Training Details

K.1 Supervised Fine-Tuning of SeekJudge-9B

We obtain SeekJudge-9B by full-parameter supervised fine-tuning of the language model of Qwen3.5-9B, keeping the vision encoder and the vision-language aligner frozen. The training set holds 45,416 samples and about 151M tokens per epoch, and it mixes trajectory-level judging targets, per-step analysis targets, and tool-query targets. We train for 2 epochs with ms-swift, using AdamW with $\beta = (0.9, 0.95)$, a weight decay of 0.01, a learning rate of $1e{-5}$ with a cosine schedule and a warmup ratio of 0.05, gradient clipping at 1.0, a global batch size of 32 sequences, and a maximum sequence length of 32K tokens. Training runs on $8\times$ RTX A6000 GPUs with DeepSpeed ZeRO-3 and CPU offloading and takes about 41 hours.

K.2 Reinforcement Learning Configuration

This subsection lists the RL hyperparameters and the evaluation protocol that Section 4.2 summarizes. We optimize every actor with GRPO and train one policy per domain across the two backbones UI-TARS-1.5-7B and Qwen3-VL-8B, keeping the domains separate so that each number reflects a small model specialized to a single domain. Every run uses a train batch size of 8 with a group size of 8 rollouts per prompt, a learning rate of $1e{-6}$, and an actor image-history length of 2, and runs for 75 training steps on $8\times$A100 GPUs in roughly one day. The GRPO objective uses no KL penalty, clipping ratios of 0.2 and 0.28 for the lower and upper bounds, and token-mean loss aggregation. Rollouts sample at temperature 1.0, and test evaluation decodes greedily. The Chrome, Impress, and OS domains hold 23, 24, and 12 training tasks and 23, 23, and 12 test tasks.

For test evaluation we run the policy every 15 training steps, repeating the pass three times over a held-out test set at each checkpoint. The value reported in the main results is the mean over all test evaluations collected during training. Averaging over checkpoints reduces the fluctuation of any single checkpoint. The training success rate is measured by rule-based evaluation on the training tasks and reported as an exponential moving average over training steps. Each rollout episode is capped at 15 environment steps during training and 25 during test evaluation, following ACuRL (Xue et al., 2026), with a maximum prompt length of 13K tokens and a response length of 512 tokens.

Figure 13 plots the test success of every reward at the five test evaluations, taken every 15 of the 75 training steps, averaged over the two backbones and three domains. SeekJudge stays above every other reward at all five training steps. Rule-based reward rises steadily over training, while CUAJudge and OS-Themis end below their value at step 15.

Table 13: The nine step classes, their step scores S^{step} in the RL reward, and the classes counted as correct steps in offline evaluation (Correct). The other three classes are the erroneous steps.

Class	Definition	S^{step}	Correct
`milestone`	Completes a sub-goal of the task by producing its end-state result.	$+0.15$	✓
`correct_decision`	Picks the right one of at least two visible options when the choice needs icon recognition or application knowledge rather than text matching.	$+0.10$	✓
`error_correction`	Reverses an earlier wrong action.	$+0.05$	✓
`neutral`	A step on the correct workflow that fits no other class.	0	✓
`off_path`	A step outside the correct workflow, such as a detour or a step of a wrong strategy.	-0.05	✓
`missed_correction`	Continues past a clear on-screen sign of a problem, such as an error message or an empty search result.	-0.05	✓
`wrong_action`	Performs a wrong operation whose result a later step uses or that gives a wrong final result.	-0.15	×
`meaningless`	Repeats an action whose identical result has already been observed, and no later step needs it.	-0.05	×
`false_claim`	Claims completion of an incomplete task when no earlier wrong action caused the failure.	-0.15	×

K.3 Trajectory Reward from Dimension and Step Scores

This subsection gives the implementation of the trajectory reward. The reward server returns the trajectory score S_i^{traj} from the regressor and the step score $S_{i,k}^{\text{step}}$ that the rubric assigns to every step, and Eq. 1 combines them into the trajectory reward R_i. It averages the step scores instead of summing them, keeping this term on a common scale so a longer rollout earns no extra reward for taking more steps. The weight $w_{\text{step}} = 2$ was set once before training and never tuned, and we tried no other value. A group holds the $G = 8$ rollouts of one goal, and the GRPO advantage of step k of rollout i normalizes R_i over the G trajectory rewards,

$$\hat{A}_{i,k} = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^{G})}{\text{std}(\{R_j\}_{j=1}^{G})}. \tag{4}$$

The right side carries no step index k. Every decision step of the rollout shares this one advantage, and the step scores enter it only through their mean in Eq. 1. We leave per-step advantage estimation to future work. CUAJudge and OS-Themis enter the same pipeline through their native binary verdicts and receive no such calibration.

L Step Classes and Step Scores

Table 13 defines the nine step classes that the step prompt assigns, one class per step, and lists the step score S^{step} that each class contributes to the RL reward of Eq. 1. For offline evaluation, the six classes marked correct count as correct steps and the other three count as erroneous steps. This split approximates the criterion of the human step labels in CUAStepBench, which mark a step as erroneous only when it harms the final result or a later step uses its wrong result. A detour or a missed warning that the agent later recovers from counts as correct under this criterion.

M Prompt Construction

The judging process uses three prompts, the seek prompt that explores the trajectory, the overall-trajectory prompt that scores the trajectory, and the step prompt that scores each step. We call the last two together the criteria prompt. Separating exploration from scoring lets the step prompt be reused in other frameworks unchanged.

The seek prompt drives exploration. At each round the agent either calls a tool to gather more evidence or gives its conclusion. Giving a conclusion ends the phase, and otherwise the agent keeps exploring until it reaches the round limit, and then it is forced to stop and conclude.

Once the conclusion is fixed, the criteria prompt produces the detailed evaluation. It splits into two independent scoring prompts, the overall-trajectory prompt that assigns the seven dimension scores and the step prompt that assigns a step class to each step, each under its own system prompt. The step prompt depends only on the trajectory representation rather than on the exploration context, so it attaches to any framework that lacks native step-level judgment, including CUAJudge and OS-Themis, and yields the step labels that we report for those baselines.

An untrained model, such as the SeekJudge Qwen3-VL-8B backbone or CUAJudge, runs both phases in full at inference. For the trained SeekJudge model we merge the second phase into the first during distillation, so the training data has the model output the detailed evaluation directly at the end of exploration rather than under a separate scoring prompt.

We excerpt the two criteria prompts below. The excerpts keep the structure, the definitions and the decision rules of each prompt and replace its examples and repeated clarifications with [. . .].

Overall-trajectory prompt.

Overall Trajectory Evaluation: 7-Dimension Scoring (0-100)
You are evaluating an agent's GUI task trajectory. Score each dimension 0-100 based on the goal, actions, and observable results. CRITICAL DISTINCTION: You must distinguish between the agent's PERFORMANCE and the ENVIRONMENT's PROPERTIES. [...]

1. **goal_infeasibility.** The likelihood the goal is INHERENTLY impossible in this environment, REGARDLESS of agent behavior. 0 = Goal is clearly feasible. 50 = Uncertain. 100 = Goal is impossible, required functionality/data provably absent. [...] If 1 of N sub-tasks targets a non-existent object, score proportionally. Agent failure due to mistakes does NOT increase infeasibility. [...]

2. **goal_ambiguity.** How ambiguous or underspecified the goal text is. 0 = Perfectly unambiguous; exactly one correct outcome. 50 = Some ambiguity. 100 = Extremely vague; many valid interpretations. This is about the GOAL TEXT itself, not the agent's understanding of it. [...]

3. **exploration_degree.** How appropriately the agent explored relative to the task's needs. First determine if the task REQUIRES exploration. [...] Routine UI navigation (clicking tabs, opening menus, selecting options along a known path) is NOT exploration. [...]

4. **path_redundancy.** How much unnecessary redundancy exists in the path. 0 = Most direct path possible. 50 = About half the steps were unnecessary. 100 = Path dominated by wasted steps. [...] A long trajectory is NOT necessarily redundant if most steps serve the goal. [...]

5. **positive_evidence_reliability.** How reliable the POSITIVE evidence is that the goal was completed CORRECTLY. [...] An action on the wrong target produces ZERO positive evidence for that sub-task, even if the action itself was executed flawlessly. [...] An UNVERIFIED_CLAIM is NOT positive evidence by itself. [...] Use prior analysis (tool call results) as ground truth. [...]

6. **negative_evidence_reliability.** How reliable the NEGATIVE evidence is that the goal was NOT completed correctly. [...] Negative evidence is about the OUTCOME, not the PROCESS. [...]

7. **completion_progress.** How much of the goal the agent CORRECTLY completed. [...] A sub-task performed on the wrong target does NOT count as progress. [...]

Analysis Process. Step 1: Goal Decomposition. Step 2: Sub-task Verification. Step 3: Error Detection. Step 4: Independent Scoring. Step 5: Consistency Check. [...] You MUST output exactly 7 labels. [...] All values must be integers between 0 and 100.

Step prompt.

Step-Level Analysis: 9-Category Per-Transition Classification
Before outputting results, follow this analysis process:

Step A: Goal Decomposition. List each sub-goal the agent must achieve. Each sub-goal should correspond to a user-visible outcome that directly satisfies part of the goal, NOT an intermediate implementation step. [...] For each sub-goal, identify its end-state deliverable. [...]

Step B: Completion Check. For each sub-goal, determine: was it actually completed, not just attempted? [...] The agent claiming "done" does NOT prove completion. Verify against the actual state. [...]

Step C: Correct Approach Identification. Determine what the correct workflow for this task would be. [...] For each actual transition, determine whether it is on-path or off-path. [...] If the agent's overall strategy is wrong, then ALL steps executing that wrong strategy are off-path. [...]

Step D: Error Analysis. For each failed sub-goal, trace the cause: 1. Did the agent perform a concretely wrong operation whose result was consumed by later steps? If yes, that step is a wrong_action. [...] 2. After that wrong action, did the agent successfully undo/reverse it? If yes, that corrective step is an error_correction. 3. Did the environment produce a clear, observable signal that something was wrong, and the agent continued without addressing it? If yes, that step is a missed_correction. 4. If no wrong operation was consumed downstream, but the agent claimed completion when the task was incomplete, that step is a false_claim. [...]

Step E: Achievement Analysis. 1. Which step made the sub-goal transition from not-done to done? That step is a milestone. [...] 2. At any step, did the agent select correctly from $\geq$2 visually present alternatives, where the correct option required non-textual reasoning (icon recognition, indirect navigation, domain knowledge)? That step is a correct_decision. [...]

Step F: Remaining Steps. 1. Does this transition REVERSE progress toward a correct state? If yes, off_path. [...] 2. Is the action an exact repeat with identical result already observed AND not causally necessary? If yes, meaningless. 3. Is this step off-path (from Step C)? If yes, off_path. 4. Otherwise, neutral.

Category Definitions (9 categories, mutually-exclusive). [...]
wrong_action. Key test: Was the wrong output used downstream, or is this the final step that produces an incorrect result? [...]
false_claim. Key test: Is it the done-report AND is the failure from omission (not from a prior wrong_action)? [...] Only wrong_action blocks false_claim. [...]
error_correction. Key test: Does it explicitly undo a specific prior error?
missed_correction. Key test: Did the environment produce an objectively visible signal of a problem, and the agent proceeded without correcting it? [...] Only mark the most salient signal. [...]
milestone. Key test: Can I point to which sub-goal was completed? Is this the step that produces the end-state deliverable, not a preparatory step? [...]
correct_decision. Key test: Are there $\geq$2 plausible options visible AND the agent picked right AND a text-only user couldn't make this choice? [...]
off_path. Key test: Would an ideal agent who knows the correct approach have taken this step? If no, off_path. [...]
meaningless. Key test: Was this exact action+result already seen before? [...]

```
neutral.  Key test:  Is this step on the correct workflow path AND
does it not fit any other category?

[...]  Each transition gets exactly ONE category.  [...]  You MUST
output exactly one line per transition, covering ALL transitions from
Transition 1 to the last transition in the trajectory.  [...]
```