

When the Curriculum Writes the User: Request Language and Reward in ACuRL

nullreference

Abstract

ACuRL trains computer-use agents on model-generated tasks and scores their trajectories with CUAJudge. This couples the training request distribution to the generator and the reward to a learned evaluator. We audit all 9,216 public curriculum tasks, rate matched prompt samples, run the released Writer agent on three versions of one OSWorld task, and inspect the public reward parser. One separate model rated all 120 sampled ACuRL prompts as highly specific. In the GUI trials, the ACuRL checkpoint passed the exact task, but it did not save a conversational paraphrase within 12 turns. It also chose unrequested formatting for a vague request. A public CUAJudge parser probe returned reward for the label “unsuccessful.” These findings identify testable risks. They do not measure ordinary user language, general agent transfer, or an exploited judge.

1 Claim and scope

Autonomous curriculum reinforcement learning creates tasks from environment experience and agent feedback. ACuRL uses this loop to adapt computer-use agents, then scores trajectories with CUAJudge [1]. The paper reports gains across six environments, cross-environment learning, and recovery under software and interface changes. It also reports human checks of generated tasks and evaluator judgments.

The open question is whose language the curriculum represents. A task-writing model can produce clear requests in a style that resembles its own. A general-purpose assistant must also handle short, indirect, or incomplete requests. A user who says “make this look better” may expect a question, and may accept several different results. A generator’s ability to produce executable requests does not show that its tasks match human requests.

The evidence below does not settle this question. It includes a small matched agent trial, not a benchmark replication. Benchmark success, judge agreement, and user satisfaction measure different outcomes.

2 What the paper establishes

ACuRL evaluates trained agents on tasks from OSWorld, OfficeWorld, and ScienceBoard. It reports average results over three runs. Its experiments include six environments, cross-environment sequences, changes to software, platform, and screen resolution, and ablations of iterative training and curriculum generation. The reported gains therefore extend beyond the generated training tasks [1].

The paper also tests CUAJudge against human judgments. Reviewers assessed 288 trajectories across six environments and three training iterations. The paper reports 93.7% overall agreement, 94.5% precision, and 93.8% recall. Agreement by iteration is 96.6%, 89.1%, and 95.6%. Reviewers saw full trajectories, including actions and screenshots. They sent uncertain cases to a second reviewer for discussion [1].

ACuRL also reports a human review of 144 generated tasks. Reviewers rated 94% of them meaningful and executable. This supports task validity. It does not measure coverage of user requests or whether prompts admit multiple acceptable outcomes [1].

The released CUAJudge implementation uses models to extract task key points, rate trajectory screenshots, and make a final success decision. The paper’s human comparison supports the full pipeline on its sampled trajectories. It does not test every stage by task type or test the reward parser with malformed judge outputs [3].

These checks provide useful evidence. The concern is not that the authors failed to test transfer or human alignment. The tests use bounded benchmark and trajectory samples. The reported agreement does not test whether the policy later learns to exploit rare judge errors. Average agreement also does not show how labels vary with prompt ambiguity, error type, or user preference.

The generator and judge shape different parts of the same loop. The generator sets the request language and which tasks appear. The judge decides which outcomes earn reward. A judge that can score whether an office action

occurred may still miss broken code, weak retrieval evidence, poor writing, or unwanted side effects. Agreement on one task family does not establish accuracy on another, or predict what a user prefers when intent is open. We do not measure those capabilities here.

3 Public task-release audit

The public ACuRL dataset contains two splits, one for Qwen3-VL and one for UI-TARS. Each split has 4,608 tasks across six environments and three iterations. GPT-5 generated tasks for both splits [2]. We read both Parquet files with DuckDB and measured exact duplicates and task length. We define task length as whitespace-separated words. We report medians and 90th percentiles without inferential tests because the release is a descriptive artifact, not a random sample of user requests.

Table 1: Task length in the public ACuRL curriculum release.

Agent	Iteration	Tasks	Median words	90th percentile
Qwen3-VL	1	1,536	13	21
Qwen3-VL	2	1,536	15	23
Qwen3-VL	3	1,536	16	24
UI-TARS	1	1,536	13	20
UI-TARS	2	1,536	15	22
UI-TARS	3	1,536	15	23

The length increase is consistent with the paper’s report that task length and execution steps grow across iterations. Fifteen rows repeat an existing task in the Qwen3-VL split, and 30 rows do so in the UI-TARS split. Sixteen distinct task texts appear in both splits. These counts describe text reuse. They do not measure semantic diversity or task coverage.

This audit describes two agent splits, six environments, three iterations, and one task generator. The release has no separate set of user-authored ambiguous requests. Benchmark results provide transfer evidence, but they do not directly measure how an agent handles a request with missing details or several acceptable outcomes.

4 Pilot comparison of task wording

We compared a balanced sample of ACuRL curriculum instructions with tasks from OSWorld. The ACuRL split contains tasks written by GPT-5 for the Qwen3-VL and UI-TARS curricula [2]. OSWorld contains 369 benchmark tasks based on real-world computer-use cases. Its prompts are curated benchmark instructions, not transcripts of ordinary user requests [4].

We sampled 15 tasks per application from each source for LibreOffice Calc, LibreOffice Impress, LibreOffice Writer, and Thunderbird. These applications appear in both datasets. This gave 60 tasks from each ACuRL split and 60 OSWorld tasks. A fixed seed made the sample reproducible. We hid the source label and asked a separately downloaded Qwen3.5-4B model [5] to rate specificity and the amount of unstated inference on scales from 1 to 5. It also rated whether an agent should ask for clarification, might ask, or could act. The model ran in bfloat16 on the NVIDIA GB10. Peak allocated CUDA memory was 10.36 GiB.

Table 2: Blind ratings of sampled task wording by Qwen3.5-4B.

Task source	Tasks	Median specificity	Median inference	Ask to clarify
ACuRL, Qwen3-VL split	60	5	1	0/60
ACuRL, UI-TARS split	60	5	1	0/60
OSWorld	60	4	2	14/60

Under this rubric, the model rated every sampled ACuRL request as highly specific and low in inference. It marked 14 OSWorld requests as needing clarification. The four application groups showed the same direction. This supports a limited observation: the sampled curriculum prompts read as more fully specified to this separate model than the matched OSWorld prompts.

This pilot does not establish that OSWorld represents general user language. OSWorld tasks are curated benchmark instructions, and one model supplied all ratings without human validation. The score difference may reflect task design as well as author voice. It does not show that ACuRL agents would fail on ordinary requests. It gives a concrete comparison that the released curriculum alone cannot provide, and a reason to test the same user goal under different wording.

5 Matched Writer task

We ran the released Qwen3-VL-8B base checkpoint and the ACuRL Writer checkpoint on the same OSWorld LibreOffice Writer task. We kept the starting document and GUI agent fixed. We changed only the request. The exact request asks the agent to center-align the heading. The paraphrase asks it to put the title in the middle of the page. The vague request asks it to make the heading look better. Each run started from a fresh VM. We recorded screenshots, model responses, actions, and the saved DOCX. The benchmark checker tests alignment of the first paragraph in that saved file.

The exact task passed for both checkpoints. The ACuRL checkpoint terminated after four model turns. The base checkpoint reached the same saved state but repeated the center-align action through the 12-turn limit. The paraphrase split the outcomes: the base checkpoint passed and terminated after three turns. The ACuRL checkpoint did not pass after 12 turns. In its separate six-turn run, a screenshot showed the heading centered in the live document. The saved DOCX still failed the alignment check. In the 12-turn repeat, the agent left a paragraph dialog open and did not terminate.

Table 3: One OSWorld Writer task under three request wordings.

Request	Qwen3-VL-8B base	ACuRL Writer
Benchmark wording	Saved check 1. Twelve turns. No termination.	Saved check 1. Four turns. Terminated.
Paraphrase	Saved check 1. Three turns. Terminated.	Saved check 0. Twelve turns. No termination.
Vague request	No target score. Guessed a style. No termination.	No target score. Guessed a style. No termination.

Both agents acted on the vague request without asking for clarification. The ACuRL checkpoint opened character formatting and selected a bright red font color in its six-turn run. In the 12-turn repeats, both checkpoints changed paragraph settings in the interface, but neither saved a change to the document. These screenshots record possible choices, not user-approved outcomes. The agent tool schema offers GUI actions and termination, but no action for asking the user.

These runs do not estimate transfer rates. They use one task, one paraphrase, one vague request, and two checkpoints. They show that the same trained agent can pass a benchmark wording and fail to save the same target under a conversational paraphrase. They also show that an underspecified request can trigger an arbitrary formatting choice. They do not show what ordinary users would prefer.

6 Why reward agreement leaves an open question

CUAJudge extracts key points from a task and evaluates a trajectory against them. This makes a generated task with explicit state changes well suited to a completion check. A request such as “make this slide look better” does not define a single expected state. A user may value readability, brand consistency, or a particular visual style. A judge can verify stated criteria only after someone supplies them.

This example does not imply that ACuRL trains on vague requests or that CUAJudge mislabels them. It identifies a construct boundary. Existing results measure completion under specified tasks. They do not measure how agents handle requests whose intended outcome needs clarification or admits several valid results.

Agreement on a fixed validation sample is different from reward reliability over repeated policy optimization. If a policy receives reward from a judge over many training attempts, rare judge errors can become useful signals. This is a general risk of optimizing against a learned evaluator, not a reported ACuRL failure. The human sample across iterations is a meaningful check, but it does not estimate error rates on policy-selected failures or on an unfamiliar request distribution.

The public reward parser has a concrete false-positive condition. It assigns reward 1 when the text after “Status:” contains the substring “success.” We ran this method from the public code revision and found that it accepts “Status: unsuccessful” and “Status: failure, not successful.” The second case can receive reward 1 even though its status says failure. This probe tests the parser with crafted output. It does not show that an ACuRL policy caused a judge to produce that output.

Table 4: CUAJudge reward-parser probe on output strings.

Text after the status label	Reward
failure	0
unsuccessful	1
failure, not successful	1

Key-point completion can also miss unwanted side effects. An agent may reach the stated goal while changing unrelated content, overwriting a file, or ignoring a preference that the request did not spell out. These are possible forms of reward mismatch. We did not observe a saved side effect in the vague Writer trials, and we did not test a policy-level reward exploit.

7 A broader test

An evaluation should hold out requests written by users who did not write the benchmark or curriculum prompts. For each computer task, collect an explicit request, a normal conversational version, an underspecified version, and a version with more than one acceptable outcome. Have users define acceptable outcomes before the agent runs. Let the agent ask a clarifying question when the request lacks enough information. Keep the initial computer state and intended goal fixed across wording variants.

Compare the base agent with ACuRL-trained agents at the same model scale. Test both on existing benchmark tasks and on held-out user requests. Measure task completion, clarification, user satisfaction, and unrelated state changes. Report results by task type and environment. Use blinded reviewers and report uncertainty across tasks and reviewers.

To test judge quality, build held-out examples for retrieval, coding, office editing, and writing. Ask human reviewers to label goal completion, factual or functional quality, and unintended changes. Compare those labels with CUAJudge before and after policy optimization. Measure false positives and false negatives by task type. Also test malformed judge outputs against the reward parser, then use adversarial tasks to test whether a policy can cause those outputs. To test generator effects, train matched policies with the original generator and an independently prompted generator. Keep the training budget fixed and evaluate both on the same held-out user requests.

The concern would gain support if benchmark gains fail to transfer, if satisfaction falls while judge reward rises, or if ratings change sharply under a generator or judge swap. It would weaken if ACuRL improves both task completion and user satisfaction across clear, ambiguous, and multi-outcome requests, with human judgments tracking CUAJudge after optimization.

8 Limits

This note analyzes the paper, released task text, one model-rated prompt sample, one OSWorld task, and the public reward parser. It does not rerun ACuRL training or CUAJudge, and it does not test a policy-level reward exploit. The language pilot uses one rater without human labels. The GUI trial uses one task and three prompts. It cannot establish population language, user satisfaction, or general agent performance.

9 Conclusion

ACuRL reports benchmark gains and high agreement between CUAJudge and human reviewers on sampled trajectories. Our one-task trial found that the ACuRL checkpoint passed the benchmark wording and failed to save a conversational paraphrase. The base checkpoint showed the reverse pattern. Both agents guessed at a vague request and failed to stop. These examples do not estimate general performance or user preference. The reward

parser also has a source-level false-positive condition, but we did not show that a policy exploited it. Human-written tasks and post-optimization judge checks remain necessary to measure how well curriculum gains match user goals.

Reproducibility

The audit scripts, sampled model ratings, GUI trajectories, parser probe, and run instructions are in the separate ‘acurl-audit’ code workspace. The task-count audit uses DuckDB 1.5.5. The language pilot used Qwen3.5-4B from revision `851bf6e8` with Transformers 5.3.0 and PyTorch 2.10.0+cu130. It ran on an NVIDIA GB10 with CUDA. The GUI trial used Qwen3-VL-8B-Instruct revision `0c351dd0` and ACuRL Writer revision `020174a1`, served on the same NVIDIA GB10. Both checkpoints were fetched at the pinned revisions for these runs. The trial used ACuRL’s Qwen3VLAgent and OSWorld commit `b138d348`, task `3ef2b351`, and the Docker provider. We used maximum turn limits of six for initial trials and 12 for paraphrase and vague prompt repeats. The saved-file check used OSWorld’s first-paragraph alignment rule. The Docker provider required a local boot image URL in this setup. The small patch and exact task-specific actions, screenshots, and results are in the code workspace. The CUAJudge parser probe reads the method from public code revision `bc57efbd`. The language script validates each sample size and JSON rating.

References

- [1] T. Xue, Z. Liao, T. Shi, Z. Wang, K. Zhang, D. Song, Y. Su, and H. Sun. Autonomous Continual Learning for Environment Adaptation of Computer-Use Agents. arXiv:2602.10356, version 2, 2026. <https://arxiv.org/abs/2602.10356>.
- [2] OSU NLP Group. ACuRL Curriculum Tasks. Hugging Face Datasets, 2026. <https://huggingface.co/datasets/osunlp/ACuRL>.
- [3] OSU NLP Group. ACuRL reference implementation, commit `bc57efbd77a8bb3d2bfc67d1aa701c20c1864d91`. GitHub, 2026. [Pinned CUAJudge source file](#).
- [4] T. Xie et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. arXiv:2404.07972, 2024. <https://arxiv.org/abs/2404.07972>.
- [5] Qwen Team. Qwen3.5-4B. Hugging Face, 2026. <https://huggingface.co/Qwen/Qwen3.5-4B>.