SCALECUA: Scaling Computer Use Agents with Verifiable Task Synthesis and Efficient Online RL

Bowen Lv[1*†], **Xiao Liu**[1,2*], **Yanyu Ren**[1†], **Hanyu Lai**[1†], **Bohao Jing**[2†], **Hanchen Zhang**[1†], **Yanxiao Zhao**[1†], **Shuntian Yao**[1†], **Jie Tang**[1], **Yuxiao Dong**[1]

[1]Tsinghua University [2]Z.AI

Abstract

Computer use agents (CUAs) are emerging as a powerful interface for automating complex digital workflows through visual perception and GUI execution. Online reinforcement learning with verifiable rewards (RLVR) has emerged as a key direction for scaling their capabilities. However, applying this paradigm is severely bottlenecked by verifiable data scarcity and online RL inefficiency. To break these barriers, we introduce **SCALECUA**, a unified framework that scales online RL for CUAs by combining verifiable task synthesis with efficient online RL. At the data level, we design **VERIGEN**, an end-to-end framework for generating verifiable RL tasks through iterative docker interactions and a multi-agent feedback loop. Scaled to 100+ concurrent agent workers via a shared docker interaction probe, this pipeline produces **24K+** verifiable tasks and nearly **3K** high-quality RL tasks for online RL. To maximize sample efficiency over this scaled pool, we propose **Frontier Sampling**, which dynamically tracks the model's per-task capability and allocates rollouts to tasks at the current learning frontier. On the training side, we further design Visual Context Segmentation, which keeps the recent visual context within a sliding window to balance rollout and training engine pressure on long-horizon trajectories, yielding a **2.83**× training speedup over step-wise decomposition. Together, SCALECUA achieves **68.7%** on OSWorld and **54.0%** on ScienceBoard, establishing new state-of-the-art performance among open-source computer use agents. Source code is available at `https://github.com/THUDM/SCALE-CUA`.

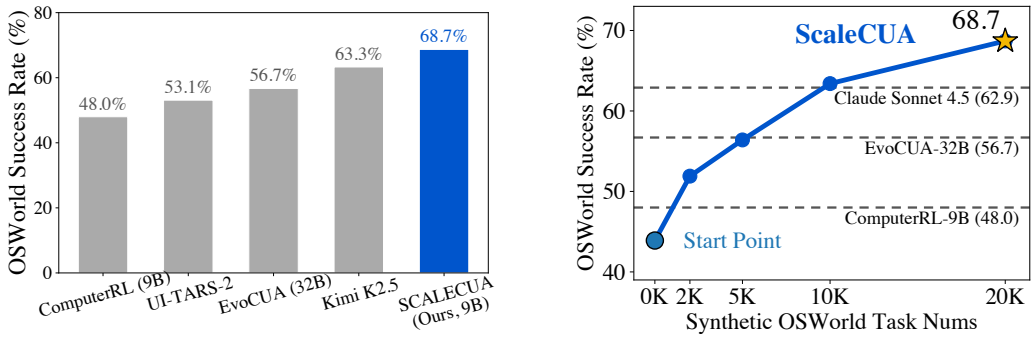

Figure 1: **Left:** Comparison with open-source models on OSWorld. SCALECUA achieves 68.7%, surpassing most prior open-source models including those 4× larger. **Right:** Effect of scaling generated verifiable tasks on OSWorld performance under the same training pipeline.

*Equal contribution. †Work done while these authors interned at Z.AI.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Computer use agents (CUAs) have emerged as a transformative interface for automating everyday digital workflows through visual perception and GUI action execution [43, 54, 12, 22]. Unlike traditional automation built on structured APIs, CUAs operate directly on screen pixels and UI elements, enabling profound generalization across diverse applications and operating systems [10, 40, 26]. Each CUA task is naturally a multi-turn interaction loop: the agent observes a screenshot, acts on the interface, and iterates until task completion. Recent commercial breakthroughs such as Claude Computer Use [2] and OpenAI Operator [24] demonstrate the immense practical value of this paradigm. To further push the boundaries of CUA capabilities, the research community has begun shifting toward online reinforcement learning with verifiable rewards (RLVR) [4, 25, 20, 45, 21].

However, while RLVR has shown initial promise, scaling it for CUAs remains largely unsolved due to verifiable data scarcity and online RL inefficiency. On the data side, the bottleneck lies in producing verifiable and suitable GUI tasks at scale. A verifiable GUI task is one paired with a deterministic judge function (e.g., a filesystem check, a browser-state inspection, or a piece of Python code) that scores the final environment state without any human or LLM judgment. Unlike math [30, 11, 49] or code [16], where verifiers can be obtained easily, a GUI task naturally consists of only a live environment and an initial instruction, without a built-in mechanism to score the resulting state. This makes scaling verifiable GUI tasks fundamentally challenging. A suitable GUI task is one that is neither trivially easy nor out-of-reach for the current model. Under GRPO-style training, tasks where the model always succeeds or always fails contribute zero within-group advantage and waste the entire rollout group, so randomly synthesized tasks easily produce wasted rollouts at scale. Prior work on automated GUI task synthesis has not addressed both axes—most efforts focus on trajectory-level data that is neither verifiable nor difficulty-controlled.

On the online RL side, inefficiency emerges in rollout sampling and multi-turn training. In terms of sampling, uniform task selection under GRPO-style training [30] fails at both the per-batch and per-task level. At the per-batch level, uniform sampling frequently produces rollout groups in which all tasks either succeed or fail, which provides no useful signal for the RL training [52]. At the per-task level, the model cannot quickly handle specific tasks, as the sampling frequency of these tasks decreases when the task pool scales up. In terms of training, the multi-turn GUI setting faces a rollout-training trade-off in how each rollout trajectory is packaged into training items. Trajectory-level training keeps the entire trajectory as a single training item, which accumulates all interaction screenshots but severely slows the rollout engine; the resulting long sequences also stress the training engine. Step-wise training instead emits one training item per interaction turn, each retaining only the final screenshot. This relieves the rollout engine, but the number of training items grows linearly with the average number of interaction turns, substantially slowing overall training.

To systematically break these barriers, we present **SCALECUA**, a unified framework for scaling computer use agents that combines verifiable task synthesis with efficient online RL. For verifiable task synthesis, we design **VERIGEN**, an end-to-end pipeline that takes both environment knowledge documents and model rollout trajectories as input, and iteratively interacts with live docker environments through a multi-agent feedback loop to refine each task and judge. For efficient sampling, we propose **Frontier Sampling**, which continuously tracks the sampling weight of each task and dynamically selects a batch of tasks best matched to the model's current capability before each rollout. For multi-turn training, we propose **Visual Context Segmentation**, which keeps the recent visual context within a sliding window while preserving textual continuity. This controls visual-token growth during long-horizon training and avoids the sample-count inflation of step-wise decomposition, enabling more efficient optimization over large-scale multi-turn rollouts.

We apply SCALECUA to several leading vision-language models, including GLM-4.6V-Flash [13], Qwen3-VL-8B-Thinking [5], and Qwen3.5-9B [48], observing consistent gains across all three models. With Qwen3.5-9B, SCALECUA achieves **68.7%** on OSWorld [43] and **54.0%** on ScienceBoard [34], establishing new state-of-the-art results among open-source computer use agents. Our 9B model surpasses prior open-source baselines on OSWorld, including ComputerRL-9B (48.0%), EvoCUA-32B (56.7%), and Kimi K2.5 (63.3%).

In summary, our main contributions are:

- We present VERIGEN, an end-to-end framework that generates verifiable and suitable GUI tasks through iterative docker interactions and a multi-agent feedback loop. Through a shared docker

Table 1: Comparison of representative GUI-agent methods along the scaling dimensions.

Method	Verifiable Task Scaling	Efficient Sampling	Large-scale Online RL	Efficient Multi-turn Training
GUI-Genesis [7]	✓	✗	✗	✗
OS-Genesis [33]	✗	✗	✗	✗
OpenCUA [39]	✗	✗	✗	✗
ComputerRL [20]	✗	✗	✓	✗
DART [21]	✗	✓	✓	✗
MobileRL [45]	✗	✓	✓	✗
Mobile-Agent-v3.5 [44]	✗	✓	✓	✗
ARPO [9]	✗	✗	✗	✗
EvoCUA [47]	✓	✗	✗	✗
SCALECUA	✓	✓	✓	✓

interaction probe, VERIGEN scales to 100+ agent workers and 100+ docker environments concurrently, yielding **24K+** verifiable tasks and nearly **3K** high-quality RL tasks that establish the data foundation for SCALECUA.

- We introduce Frontier Sampling, a lightweight data sampler that maintains per-task sampling weights and selects training tasks aligned with the model's current capability before each rollout. With small per-step overhead, it provides efficient learning signal in every batch and speeds up the model's progression from easier to harder tasks.
- We propose Visual Context Segmentation, a sliding-window strategy that bounds visual-token accumulation while preserving textual continuity, balancing the pressure between rollout and training engines. This achieves a **2.83**× end-to-end training speedup over step-wise decomposition without sacrificing cross-step reasoning.

2 Related Work

Computer Use Benchmarks. Computer use agent evaluation has evolved from web navigation benchmarks such as Mind2Web [8] and WebArena [54] to full desktop environments such as WindowsAgentArena [6], OSWorld [43], and ScienceBoard [34]. OSWorld evaluates open-ended desktop tasks in real operating systems, while ScienceBoard focuses on scientific workflows involving professional software and domain-specific operations.

Data Generation for GUI Agents. Recent systems expand agent data through task or trajectory synthesis, including AgentTrek [46], AgentGen [15], AndroidGen [19], OpenCUA [39], OS-Genesis [33], GUI-Genesis [7], and AgentSynth [41]. These works reduce manual data construction, but many focus on reasoning trajectories, task proposals, mobile/web environments, or reconstructed lightweight settings rather than executable judge synthesis in full desktop OS environments. GUI-Genesis is closely related in its use of code-native verifiable rewards, yet it generates tasks in reconstructed lightweight web environments. VERIGEN addresses this gap with an end-to-end framework that synthesizes verifiable GUI tasks through iterative interaction with live OS containers, scaling task generation far beyond manually curated pools.

Online RL for GUI Agents. Online RL for GUI agents was pioneered by DigiRL [4] in live Android environments, then extended to web tasks by WebRL [25] with curriculum-based task selection. Subsequent work brought online RL to desktop and mobile platforms, including ARPO [9], DART [21], ComputerRL [20], EvoCUA [47], MobileRL [45], and multi-platform systems [31, 51, 44], demonstrating that executable feedback can improve computer use models across environments. However, these methods remain limited in scaling. Uniform sampling and fixed curricula waste expensive rollouts on tasks that are already solved or out of reach, while step-wise decomposition multiplies training items with interaction length, slowing the training engine [20, 21, 45]. SCALECUA instead addresses these inefficiencies through Frontier Sampling for upstream task allocation and Visual Context Segmentation for balanced multi-turn training.

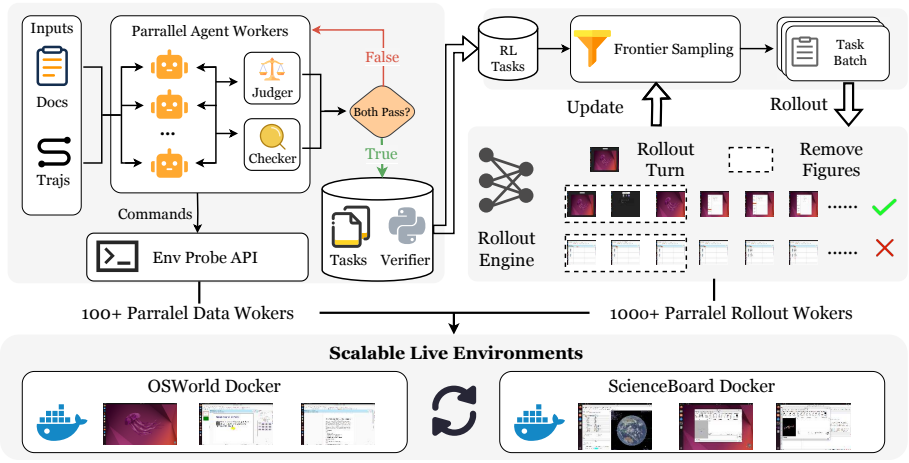

Figure 2: Overview of SCALECUA. VERIGEN autonomously synthesizes verifiable GUI tasks with executable rewards from live OS environments at 100+ parallel worker scale; combined with upstream Frontier Sampling and long-horizon Visual Context Segmentation, SCALECUA enables efficient large-scale online RL for computer use agents.

3 The SCALECUA Framework

3.1 Overview

SCALECUA is a unified framework that combines verifiable task generation and efficient online RL to scale computer use agents. On the data side, VERIGEN (§3.2) synthesizes verifiable GUI tasks at scale through iterative interaction with live OS environments. On the RL side, Frontier Sampling (§3.3) lightly tracks each task's sampling weight to direct rollouts at the model's current frontier, while Visual Context Segmentation (§3.4) balances rollout and training engine pressure through sliding-window trajectory processing.

Concretely, the SCALECUA pipeline runs in three stages. First, we train the base model on the initial verifiable task pool generated by VERIGEN. Next, we collect the trained model's rollouts on a subset of these tasks, and VERIGEN applies trajectory-guided synthesis to convert these rollouts into a refined RL-ready task pool. Finally, we run online RL on this refined pool with Frontier Sampling and Visual Context Segmentation.

3.2 VERIGEN: Scaling Verifiable Task Pools

To solve the scarcity of verifiable GUI tasks, we build VERIGEN, an end-to-end framework that synthesizes them by driving LLMs to interact with live OS containers and validating each judge through real execution. Appendix B.1 shows an example of a generated task.

To produce verifiable GUI tasks, we build an iterative framework that combines environment interaction with a multi-agent feedback loop. Three independent LLM agents—proposer, judger, and checker—play distinct roles in this loop. Starting from knowledge documents extracted per environment and a live docker container interface, the proposer agent generates an initial synthetic task with its verifier. On one path, an independent judger agent performs a static semantic review to evaluate whether the task instruction is unambiguous and meaningful and whether the verifier's logic is theoretically sound, returning a verdict with reasoning. On the other path, an independent checker agent runs a dynamic check by performing a "dry run" within the docker environment: it actively executes actions and observes the resulting state updates (screenshots and accessibility trees) to test whether the task goal is physically reachable and whether the verifier correctly evaluates the final outcome. Combining both signals, the framework produces a final judgment on the candidate and decides whether to feed it back into the next loop.

To get more suitable tasks for the model's RL training, we further extend the framework with trajectory-guided task synthesis, taking the trained model's rollout trajectories and per-task pass rates as additional input. For failed tasks, VERIGEN inspects each step's action and the environment feedback to dynamically locate the earliest step at which the model starts to deviate from the original goal. It then splits the original task into smaller subtasks, using each intermediate sub-goal as the target of a new, easier task with its own setup state and verifier. For succeeded tasks, VERIGEN clusters semantically similar tasks within each application (e.g., tasks operating on the same Excel sheet) and composes them into harder multi-objective tasks by chaining their goals and verifiers into a single instruction.

To support large-scale parallel data synthesis, we design a shared docker interaction probe that exposes an environment interface to agent workers. Through this probe, agent workers retrieve screenshots and accessibility-tree information from the docker without entering it, while the probe executes dispatched actions inside the environment. This separation lets VERIGEN run 100+ agent workers concurrently against 100+ docker containers. Combining iterative interaction, trajectory-guided synthesis, and the parallel probe, the full VERIGEN framework produces **24K+** verifiable tasks and nearly **3K** high-quality RL tasks.

3.3 Frontier Sampling over Large Task Pools

Given the scaled task pool from VERIGEN, online RL needs a more efficient sampling method for the model to converge faster. We thus propose Frontier Sampling, a lightweight data sampler that maintains per-task sampling weights and dynamically allocates rollouts to tasks at the model's current capability frontier.

To track the model's capability on each task, we maintain an exponential moving average (EMA) of per-task success rates, which updates smoothly across iterations. Before RL begins, we rollout the model on all tasks to obtain initial pass rates ($\hat{p}_i$) and filter out tasks outside $[p_{\min}, p_{\max}]$ (e.g., $[0.1, 0.9]$). Each remaining task i then has its EMA success rate $\hat{p}_i$ updated as:

$$\hat{p}_i \leftarrow (1-\alpha)\,\hat{p}_i + \alpha \cdot \tfrac{1}{n}\textstyle\sum_{j=1}^{n} \mathbb{1}[r_{i,j}=1]\,, \tag{1}$$

where n rollouts are averaged after each RL iteration and α is the EMA smoothing factor (e.g., α=0.2). To convert these EMA rates into sampling probabilities in an appropriate way, we apply a Gaussian kernel distribution. The Gaussian peaks at a target success rate, decays symmetrically on both sides, and drops off exponentially with distance, naturally directing rollouts to tasks at the model's current capability frontier. Specifically, we compute the sampling weight w_i as:

$$w_i = \exp\Big(-\frac{(\hat{p}_i-\mu)^2}{2\sigma^2}\Big), \tag{2}$$

Here μ specifies the target success rate and σ controls the bandwidth around it; we use μ=0.5 and σ=0.25 in all experiments. To preserve exploration, we additionally reserve a small fraction γ (γ = 0.2) of each batch for uniform sampling from the remaining tasks. In practice, Frontier Sampling concentrates rollouts on tasks with the most useful learning signal, accelerating model convergence over uniform sampling and curriculum-style baselines.

3.4 Visual Context Segmentation for Long-Horizon Multimodal RL

Scaling online RL for CUAs critically depends on training efficiency. In the multi-turn GUI setting, keeping the full visual sequence places a heavy burden on the rollout engine and slows inference dramatically, while splitting at every screenshot inflates training items and slows parameter updates. To balance this trade-off, we propose Visual Context Segmentation, which keeps the recent visual context within a sliding window while preserving textual continuity. This simultaneously bounds the per-rollout visual-token load on the rollout engine and the training-item count on the training engine, yielding an efficient multi-turn training scheme without sacrificing cross-step reasoning.

Consider a training batch of n trajectories with an average length of T interaction turns. We define K as the number of retained screenshots (the window size) and Δ as the minimum removal threshold (normally K=Δ). The naive trajectory-level mode produces $O(n)$ training items and step-wise decomposition produces $O(nT)$, while Visual Context Segmentation produces $O(nT/K)$.

Algorithm 1: Sliding-Window Trajectory Segmentation with Token-ID Stream

Require: Episode with T turns; K images to keep; Δ min removal threshold
Ensure: Set of trace segments $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$; $n_{\text{img}} \leftarrow 0$; tokenize $\mathcal{H} \rightarrow (\mathbf{ids}, \mathbf{mask{=}0}, \mathbf{logp{=}0})$
for $t = 1, \ldots, T$ **do**
  Sample $(\mathbf{ids}_t^{\text{gen}}, \mathbf{logp}_t^{\text{gen}}) \sim \pi_\theta$; append with $\mathbf{mask{=}1}$
  Execute action, receive o_t; $n_{\text{img}} \leftarrow n_{\text{img}} + 1$
  if $n_{\text{img}} > K + \Delta$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\mathbf{ids}, \mathbf{mask}, \mathbf{logp})\}$; drop oldest Δ images from $\mathcal{H}$
    Re-tokenize $\mathcal{H} \rightarrow (\mathbf{ids}, \mathbf{mask{=}0}, \mathbf{logp{=}0})$
  else
    $\mathbf{diff} \leftarrow \textsc{Tokenize}(\mathcal{H} \oplus o_t)[|\textsc{Tokenize}(\mathcal{H})|:]$
    $\mathbf{ids} \mathrel{+}= \mathbf{diff}$; $\mathbf{mask} \mathrel{+}= \mathbf{0}$; $\mathbf{logp} \mathrel{+}= \mathbf{0}$
  end if
end for
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(\mathbf{ids}, \mathbf{mask}, \mathbf{logp})\}$; assign reward r to all segments
return $\mathcal{S}$

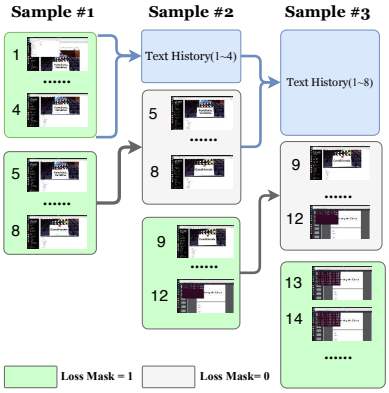

Figure 3: Visual Context Segmentation. **Left:** sliding-window trajectory processing with a token-ID stream. **Right:** when the visual window overflows, the oldest screenshots are dropped and replaced by a text summary; trainable masks (green, loss=1) cover only assistant responses, while prior tokens stay as context (gray, loss=0).

In practice, we maintain a token-ID stream (the running concatenation of input IDs, loss masks, and rollout log-probabilities) and a text history stream that preserves the full conversation with images during the rollout process. At each turn, if the text history holds fewer than $K{+}\Delta$ screenshots, the new observation and assistant response are appended to the token-ID stream via token-diff. Once the count reaches $K{+}\Delta$, pruning triggers: the current stream is saved as a training item, the oldest Δ screenshots are removed, and the remaining history is re-tokenized into a fresh token-ID stream. Figure 3 summarizes the procedure; full details of the token-ID stream are provided in Appendix C.1.

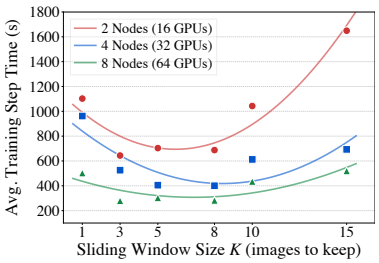

Figure 4: Sliding-Window training step time across K at 2/4/8-node scales; moderate K is optimal.

To pick K and Δ, we run a pre-experiment with all training hyperparameters fixed and sweep $K{=}\Delta \in \{1, 3, 5, 8, 10, 15\}$ across 2/4/8-node compute scales (Figure 4). Capping rollout length at 50 turns, $K{=}5$–8 reaches the local optimum at every scale, balancing per-segment visual-token pressure on the rollout engine against item-count overhead on the training engine. Under this setting, sliding-window training delivers a **2.83**× end-to-end speedup over step-wise decomposition.

4 Experiments

4.1 Setup

We apply ScaleCUA to three models—GLM-4.6V-Flash [13], Qwen3-VL-8B-Thinking [5], and Qwen3.5-9B [48]—and evaluate on OSWorld [43] and ScienceBoard [34]. Our agent operates from screenshot observations and uses each benchmark's exposed computer-action interface: OSWorld uses mouse and keyboard actions, while ScienceBoard additionally provides task-completion, answer-submission, and file-writing actions (see Appendix A for the full action space).

We build the training framework on top of AgentRL [53] and adapt it for multimodal CUA training, using vLLM [18] as the rollout engine and Megatron-LM [32] as the training engine. Related system-level scheduling optimizations for agentic RL are studied in TideRL [28]. Algorithmically, we use GRPO [30] with DAPO-style asymmetric clipping [52] and a KL penalty against the reference policy, integrating Frontier Sampling for task selection and Visual Context Segmentation for trajectory processing (full objective in Appendix D). Detailed hyperparameters and infrastructure configurations are provided in Appendix D. Tables 2–4 report benchmark-specific training; published baselines vary in harness, observation access, and turn budget. Main-table subscripts denote standard deviation

Table 2: OSWorld success rates (%). Best open-source single-model results are **bolded**; ours are shaded. Subscripts indicate the standard deviation across 4 independent rollout passes per task. †Proprietary; ‡framework with a separate planner; §overall-only results from the original authors. Due to inference cost and model availability, this comparison does not include the latest or largest variant of every model family.

Model	Steps	OS	Office	Daily	Professional	Workflow	Overall
Human	–	–	–	–	–	–	72.4
Proprietary models							
GPT-5.4 [23]†§	–	–	–	–	–	–	75.0
Claude Opus 4.6 [3]†§	–	–	–	–	–	–	72.7
Claude Sonnet 4.5 [3]†	100	70.8	72.6	61.4	63.2	49.5	62.9
Claude Sonnet 4 [3]†	50	45.8	43.5	55.1	55.1	28.5	43.9
Seed-1.8 [29]†	100	66.7	68.8	67.1	71.4	42.4	61.9
OpenAI CUA [24]†	50	70.8	23.0	37.2	51.0	15.8	31.3
CoACT-1 [14]‡	50	70.8	60.6	54.1	69.4	42.4	56.4
Agent S2.5 w/ o3 [1]‡	50	75.0	53.8	54.0	75.5	39.5	54.2
GTA1 w/ o3 [50]‡	100	62.5	54.6	60.3	61.2	38.3	53.1
Jedi-7B w/ o3 [42]‡	50	54.2	47.0	62.1	69.4	35.0	50.6
Open-source models							
Kimi K2.5 [35]	100	73.9	69.2	66.4	73.5	46.1	63.3
EvoCUA-32B [47]	50	78.3	59.8	64.5	81.6	27.9	56.7
Mobile-Agent-v3.5 [44]§	–	–	–	–	–	–	56.5
UI-TARS-2 [37]	100	41.7	61.1	62.1	61.2	34.1	53.1
ComputerRL-9B [20]	50	75.0	45.8	52.5	71.4	27.9	48.0
OpenCUA-72B [39]	50	45.8	42.7	53.8	75.5	22.3	44.9
OpenCUA-32B [39]	50	60.9	29.9	41.0	65.3	14.6	35.1
DART-GUI-7B [21]	30	62.5	39.3	50.8	73.5	16.7	42.1
Qwen3.5-9B [48]§	–	–	–	–	–	–	41.8
Qwen3-VL-8B-Thinking [5]§	–	–	–	–	–	–	33.9
Ours							
SCALECUA-GLM-4.6V-Flash†	50	**83.3**$_{\pm 4.0}$	**71.8**$_{\pm 2.7}$	64.0$_{\pm 1.9}$	85.5$_{\pm 1.1}$	46.9$_{\pm 2.4}$	66.5$_{\pm 0.5}$
SCALECUA-Qwen3-VL-8B	50	**83.3**$_{\pm 7.1}$	68.4$_{\pm 1.4}$	**79.4**$_{\pm 0.8}$	81.6$_{\pm 2.8}$	45.2$_{\pm 3.3}$	67.7$_{\pm 1.2}$
SCALECUA-Qwen3.5-9B	50	79.2$_{\pm 7.2}$	70.6$_{\pm 2.2}$	74.3$_{\pm 3.2}$	**89.5**$_{\pm 3.6}$	**48.1**$_{\pm 1.6}$	**68.7**$_{\pm 0.7}$

across evaluation passes. Transfer experiments instead use OSWorld-RL checkpoints without target-benchmark training (Section 4.6); Appendix E.1 defines their reward aggregation.

4.2 Main Results

OSWorld. Table 2 reports per-domain results on OSWorld, where we compare SCALECUA against proprietary systems (GPT-5.4, Claude Opus 4.6 / Sonnet 4.5, Seed-1.8, OpenAI CUA) and open-source baselines (Kimi K2.5, EvoCUA-32B, ComputerRL-9B, UI-TARS-2). SCALECUA-Qwen3.5-9B achieves **68.7%**, surpassing the strongest open-source model Kimi K2.5 (63.3%) and proprietary systems Claude Sonnet 4.5 (62.9%) and Seed-1.8 (61.9%), with gains most pronounced in the long-horizon Professional (**89.5%**) and Workflow (**48.1%**) categories. Qwen3.5-9B improves by **26.9** points over the officially reported 41.8% in Table 2; our measured starting point is 43.9% (Table 5).

Table 3: Recent models on OSWorld-v1 (%), using our computer tools and framework (361 tasks; eight Google Drive tasks excluded).

Model	Overall
Claude Opus 5	77.87
GPT-6 Astra	78.91
Qwen3.8-27B	63.43
GLM53-Flash	76.74
SCALECUA-GLM-5.3-Flash	78.47

Recent models. Table 3 supplements the results with OSWorld-v1 scores for several recent open-source and closed-source models, for reference.

ScienceBoard. To verify whether the pipeline remains effective on professional, knowledge-intensive software, we further evaluate on ScienceBoard [34], which spans symbolic algebra (KAlgebra), molecular visualization (ChimeraX), geographic information systems (GrassGIS), Lean theorem proving, astronomical simulation (Celestia), and scientific document authoring (TeXstudio). SCALECUA-Qwen3.5-9B reaches **54.0%** (Table 4), surpassing Claude Opus 4.6 (52.7%) and establishing a

Table 4: ScienceBoard success rates (%). Best open-source single-model results are **bolded**; ours are shaded. Subscripts indicate the standard deviation across 4 independent rollout passes per task. †Proprietary; ‡multi-agent (planner + executor).

Model	Obs.	Alg	Biochem	GIS	ATP	Astron	Doc	Overall
Human	–	74.2	69.0	55.9	42.3	51.5	68.8	60.3
Proprietary models								
Claude Opus 4.6 [3]†	screen	64.5	55.2	38.2	15.4	66.7	87.5	52.7
Claude 3.7 Sonnet [3]†	s+a11y	12.9	41.4	8.8	3.9	9.1	18.8	15.8
GPT-4o [23]†	s+a11y	22.6	37.9	2.9	7.7	3.0	12.5	14.4
Gemini-2.0-Flash†	s+a11y	16.1	24.1	2.9	0.0	18.2	12.5	12.3
o3-mini [23]†	a11y	16.1	20.7	2.9	3.9	15.2	6.2	10.8
Open-source models								
GPT-4o → GUI-Actor‡	screen	21.9	44.8	2.9	–	12.1	–	20.4
GPT-4o → Qwen2.5-VL-7B‡	screen	12.5	34.5	11.8	–	9.1	–	17.0
Qwen2.5-VL-72B	screen	22.6	27.6	5.9	0.0	9.1	12.5	12.9
Qwen3-VL-225B [5]	screen	6.5	17.2	2.9	0.0	9.1	0.0	6.5
UI-TARS-1.5 [26]	screen	12.9	13.8	0.0	0.0	6.1	0.0	5.9
InternVL3-78B	s+a11y	6.5	3.5	0.0	0.0	3.0	6.2	3.2
Ours								
SCALECUA-GLM-4.6V-Flash†	screen	$67.7_{\pm 4.6}$	$60.3_{\pm 2.8}$	**39.1**$_{\pm 6.5}$	$8.0_{\pm 4.0}$	$60.6_{\pm 3.9}$	$57.8_{\pm 5.3}$	$49.4_{\pm 2.8}$
SCALECUA-Qwen3-VL-8B	screen	**71.0**$_{\pm 6.7}$	$58.6_{\pm 2.8}$	$38.2_{\pm 2.8}$	$15.0_{\pm 3.1}$	$59.4_{\pm 3.9}$	$62.5_{\pm 13.3}$	$51.9_{\pm 1.5}$
SCALECUA-Qwen3.5-9B	screen	$67.7_{\pm 5.6}$	**62.1**$_{\pm 5.2}$	$35.3_{\pm 2.9}$	**19.0**$_{\pm 1.7}$	**60.6**$_{\pm 3.9}$	**86.7**$_{\pm 8.2}$	**54.0**$_{\pm 0.5}$

new state-of-the-art, with the largest gains on TeXstudio (**86.7%**) and Lean theorem proving (**19.0%**, vs. $\leq 15.4\%$ for all proprietary baselines), confirming that VERIGEN can synthesize verifiable tasks even for professional scientific software.

4.3 Ablation Studies

To isolate each SCALECUA component, we report OSWorld scores when individual components are removed from the full pipeline (Table 5). Starting from full SCALECUA (**68.7%**), removing VERIGEN entirely drops the model to our measured base score of 43.9%, while removing Frontier Sampling or Visual Context Segmentation alone yields 63.7% and 62.2% respectively, showing that all three components contribute substantively and complementarily.

Task-source ablation. With the RL recipe fixed, multi-agent validation/repair and trajectory guidance successively improve reward by 9.2 and 4.1 points (Table 5). Task-source rows share the RL recipe but are not task-count matched (Appendix B.10).

Table 5: **Left:** Component ablation of SCALECUA on OSWorld with Qwen3.5-9B backbone. **Middle:** Qwen3.5-9B task-source ablation: OSWorld average reward (%). **Right:** Executable rate (%) of generated judge functions when each role is removed from the VERIGEN multi-agent loop.

Configuration	OSWorld (%)
Full SCALECUA	**68.7**
w/o VERIGEN (Base)	43.9
w/o Frontier Sampling	63.7
w/o Visual Context Segmentation	62.2

Task source	Score
Single proposer + basic filtering	55.4
Multi-agent validation and repair	64.6
Full VERIGEN + trajectory guidance	**68.7**

Configuration	Exec. rate
Full pipeline	**94.5**
w/o LLM Judge Agent	62.3
w/o Fix Agent	78.1
w/o Rule Validator	86.2

VERIGEN role ablation. We isolate the contribution of the multi-agent filtering roles by removing each module from the generation loop. As shown in Table 5, the full pipeline achieves a 94.5% executable rate for generated judges. Removing the LLM Judge Agent causes the largest drop (-32.2%), followed by the Fix Agent (-16.4%) and Rule Validator (-8.3%), indicating that independent checking and repair roles are important for reliable executable reward construction.

Frontier Sampling. We compare our frontier-based sampler against three baselines: uniform sampling, DAPO [52], and curriculum learning. As shown in Figure 5(a), uniform plateaus early on already-mastered or too-difficult tasks. DAPO improves via filtered rollouts, but its allocation does not adapt to per-task capability. Curriculum learning's fixed difficulty schedule fails to track

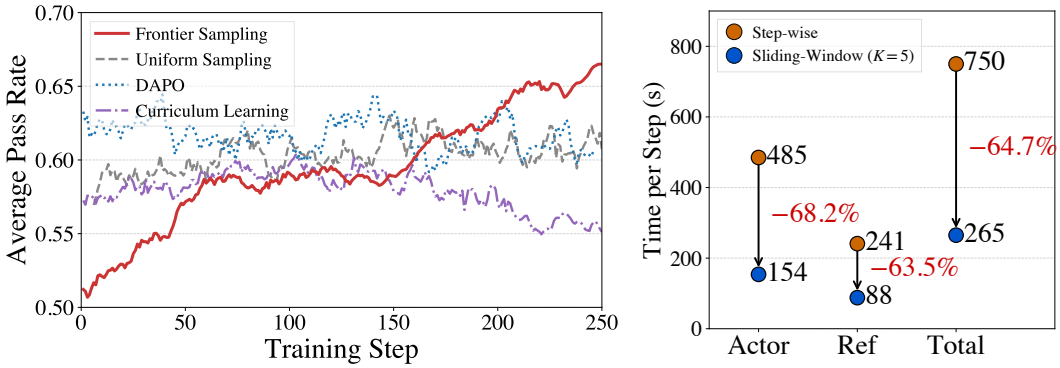

Figure 5: (a) Frontier Sampling vs. uniform, DAPO, and curriculum learning: higher reward sustained by tracking the learning boundary. (b) Sliding-Window (K=5) vs. step-wise per-phase training step time; **2.83**× end-to-end speedup.

the model's per-task success frontier. Frontier Sampling instead updates task priorities from EMA success rates and allocates more rollouts to tasks near the current learning boundary, sustaining a higher reward trajectory throughout training.

Visual Context Segmentation. We compare Visual Context Segmentation (K=5) against the standard step-wise decomposition baseline on Qwen3.5-9B with batch size 32 and rollout-n=8, holding all other hyperparameters and the compute budget fixed. As shown in Figure 5(b), it reduces the actor-update phase from 485s to 154s (−68.2%) and the reference-policy phase from 241s to 88s (−63.5%), bringing the per-step total from 750s to 265s—a **2.83**× end-to-end speedup. The savings come from collapsing $O(nT)$ step-wise items into $O(nT/K)$ Sliding-Window segments and capping per-segment visual tokens, removing the long-tail rollout latency that dominates step-wise training. We further report pass@k across $K \in \{1, 3, 5, 8, 10, 15\}$ in Appendix C.3.

4.4 Analysis of VERIGEN

Scale and coverage. With 100+ parallel workers, VERIGEN produces over 24K candidates and nearly 3K accepted RL tasks across all 10 OSWorld and 6 ScienceBoard domains, extending verifiable synthesis to professional scientific software (Appendix B.4).

Judge quality. The 160-trajectory expert audit yields **82.5%** judge–human agreement and **90.8%** verifier-positive precision; 10 false positives and 18 false negatives show that executability does not guarantee correctness. Auditing 1,874 trajectory-guided tasks finds no exact instruction duplicates but some semantic overlap with evaluation tasks; these are subset-level findings (Appendices B.7 and B.11).

Training utility. Trajectory-guided augmentation raises OSWorld performance from 64.6% to **68.7%** and sustains higher task-level reward (Figure 8, right). Turning successes into harder variants and failures into simpler subgoals provides capability-grounded variation beyond task count; details are in Appendix B.4.

4.5 Data and Training Scaling

Data scaling. We increase the training data from VERIGEN and open-source CUA task collections using the same model backbone. Both sources improve performance, while VERIGEN produces larger gains as data grow, suggesting that verifiable synthesis improves the utility of scaling task collections (Figure 6(a)).

Training scaling. We compare SCALECUA with the GRPO baseline as training progresses, expressing the optimization budget in equivalent GPU-hours. Over the observed range, SCALECUA achieves higher training rewards and stronger reward growth, demonstrating a more favorable training-scaling trend. Dashed tails illustrate extrapolations of the observed trends (Figure 6(b)).

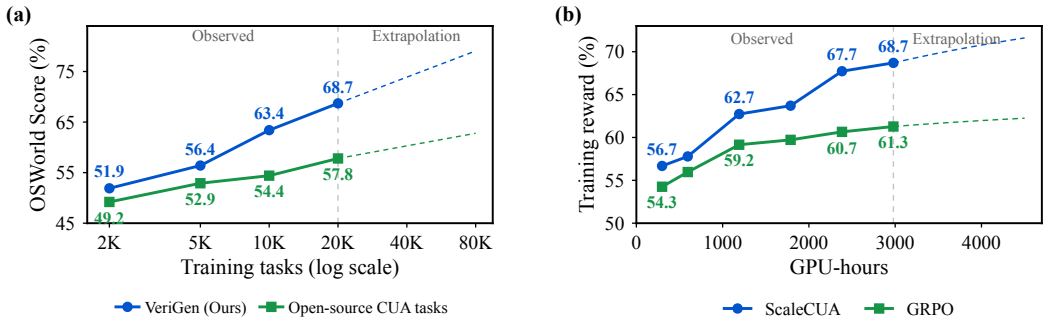

Figure 6: **Data and training-budget scaling.** (a) VERIGEN versus open-source tasks from OpenCUA/AgentNet [38] and ProCUA [17]. (b) Training reward at shared steps, with both curves converted at 11.917 GPU-hours/step using the SCALECUA run's mean step time on 64 GPUs. GPU-hours denote a unified *equivalent budget*, not GRPO's measured consumption. Dashed tails in both panels are illustrative log-linear extrapolations.

4.6 Out-of-Distribution Generalization

Table 6: OOD scores (%) before and after OSWorld RL. Bold marks the higher score in each Base/RL pair.

	ScienceBoard		OpenApps		OfficeWorld	
Backbone	Base	RL	Base	RL	Base	RL
Qwen3.5-9B	6.8	**10.1**	51.8	**59.8**	6.1	**25.6**
GLM-4.6V-Flash	5.8	**10.2**	43.8	**45.5**	4.5	**21.9**
Qwen3-VL-8B	3.8	**17.0**	**67.0**	58.9	7.1	**24.0**

Table 6 directly compares each base with its OSWorld-RL checkpoint without target-benchmark training. All three models improve on ScienceBoard and OfficeWorld [20]; on OpenApps [36], Qwen3.5 and GLM improve while Qwen3-VL declines. Full scores, gains, and evaluation protocols are in Appendices E.1 and E.2.

Model-scale pilots. Additional OSWorld pilots reach 64.8 ± 2.2 with Qwen3.5-4B and 72.1 ± 2.1 with Qwen3.6-35B-A3B (average reward $\pm$ task-level SEM). These are feasibility checks, with configurations and coverage detailed in Appendix E.3.

5 Conclusion

We presented SCALECUA, a unified framework that scales online RL for computer use agents through verifiable task synthesis and efficient online RL. On the data side, VERIGEN drives LLMs to interact with live OS containers through an iterative multi-agent loop that validates each judge by real execution. Trajectory-guided synthesis recycles rollout outcomes into new RL tasks, and a shared docker probe scales the pipeline to 100+ concurrent workers. On the online RL side, we propose Frontier Sampling to direct rollouts to tasks at the model's current capability frontier via per-task EMA success rates, and propose Visual Context Segmentation to maintain a token-ID stream with a sliding visual window, balancing rollout and training engine pressure on long-horizon multimodal trajectories. We hope SCALECUA offers a reusable recipe for verifiable-reward agent training and accelerates progress on open-source computer use agents.

Limitations. *(i)* Our experiments use bounded interaction budgets, leaving reliability on substantially longer workflows insufficiently explored. *(ii)* Our primary desktop benchmarks use Ubuntu-based environments; generalization to Windows and macOS application stacks remains to be evaluated. *(iii)* Although additional experiments extend beyond the main 8B–9B backbones, systematic studies under matched training and evaluation budgets are needed to characterize scaling across model sizes and architectures.

Acknowledgments and Disclosure of Funding

This work was supported by Tsinghua University and Zhipu AI (Z.AI).

Competing interests. Xiao Liu and Bohao Jing are affiliated with Z.AI, and Bowen Lv, Yanyu Ren, Hanyu Lai, Bohao Jing, Hanchen Zhang, Yanxiao Zhao, and Shuntian Yao conducted this work while interning at Z.AI. The authors declare no other competing interests.

References

[1] Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025.

[2] Anthropic. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku. `https://www.anthropic.com/news/3-5-models-and-computer-use`, 2024.

[3] Anthropic. Claude model card. `https://docs.anthropic.com/en/docs/about-claude/models`, 2025.

[4] 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. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

[5] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[6] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

[7] Yuan Cao, Dezhi Ran, Mengzhou Wu, Yuzhe Guo, Xin Chen, Ang Li, Gang Cao, Gong Zhi, Hao Yu, Linyi Li, et al. Gui-genesis: Automated synthesis of efficient environments with verifiable rewards for gui agent post-training. *arXiv preprint arXiv:2602.14093*, 2026.

[8] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

[9] Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025.

[10] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[12] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14281–14290, 2024.

[13] Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.5 v and glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*, 2025.

[14] Xinming Hou, Mingming Yang, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Wayne Xin Zhao. Coact: A global-local hierarchy for autonomous agent collaboration. *arXiv preprint arXiv:2406.13381*, 2024.

[15] Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pages 496–507, 2025.

[16] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The twelfth international conference on learning representations*, 2023.

[17] Jaehun Jung, Ximing Lu, Brandon Cui, Muhammad Khalifa, Shaokun Zhang, Hao Zhang, Jin Xu, Amala Sanjay Deshmukh, Karan Sapra, Andrew Tao, et al. Procua-sft technical report. *arXiv preprint arXiv:2606.17321*, 2026.

[18] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[19] Hanyu Lai, Junjie Gao, Xiao Liu, Yifan Xu, Shudan Zhang, Yuxiao Dong, and Jie Tang. Androidgen: Building an android language agent under data scarcity. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2727–2749, 2025.

[20] Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents. *arXiv preprint arXiv:2508.14040*, 2025.

[21] Pengxiang Li, Zechen Hu, Zirui Shang, Jingrong Wu, Yang Liu, Hui Liu, Zhi Gao, Chenrui Shi, Bofei Zhang, Zihao Zhang, et al. Efficient multi-turn rl for gui agents via decoupled training and adaptive data curation. *arXiv preprint arXiv:2509.23866*, 2025.

[22] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. *arXiv preprint arXiv:2408.06327*, 2024.

[23] OpenAI. GPT-5.4 system card. `https://openai.com/index/introducing-gpt-5-4/`, 2025.

[24] OpenAI. Introducing Operator. `https://openai.com/index/introducing-operator/`, 2025.

[25] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

[26] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

[27] Qwen Team. Qwen3.6-35B-A3B model card. `https://huggingface.co/Qwen/Qwen3.6-35B-A3B`, 2026.

[28] Yanyu Ren, Xizheng Wang, Xiao Liu, Bowen Lv, Hanchen Zhang, Shudan Zhang, Hanyu Lai, Shuai Wang, Li Chen, Dan Li, et al. Tiderl: Boosting agentic rl goodput with readiness-aware scheduling. *arXiv preprint arXiv:2608.10402*, 2026.

[29] Bytedance Seed. Seed1. 8 model card: Towards generalized real-world agency. *arXiv preprint arXiv:2603.20633*, 2026.

[30] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[31] Yucheng Shi, Wenhao Yu, Zaitang Li, Yonglin Wang, Hongming Zhang, Ninghao Liu, Haitao Mi, and Dong Yu. Mobilegui-rl: Advancing mobile gui agent through reinforcement learning in online environment. *arXiv preprint arXiv:2507.05720*, 2025.

[32] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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

[34] Qiushi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao, Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, et al. Scienceboard: Evaluating multimodal autonomous agents in realistic scientific workflows. *arXiv preprint arXiv:2505.19897*, 2025.

[35] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

[36] Karen Ullrich, Jingtong Su, Claudia Shi, Arjun Subramonian, Amir Bar, Ivan Evtimov, Nikolaos Tsilivis, Randall Balestriero, Julia Kempe, and Mark Ibrahim. Openapps: Simulating environment variations to measure ui-agent reliability. *arXiv preprint arXiv:2511.20766*, 2025.

[37] Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025.

[38] Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, et al. Opencua: Open foundations for computer-use agents. *Advances in Neural Information Processing Systems*, 38:139756–139806, 2026.

[39] 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. *arXiv preprint arXiv:2508.09123*, 2025.

[40] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

[41] Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents. *arXiv preprint arXiv:2506.14205*, 2025.

[42] Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, et al. Scaling computer-use grounding via user interface decomposition and synthesis. *arXiv preprint arXiv:2505.13227*, 2025.

[43] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

[44] Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026.

[45] Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. Mobilerl: Online agentic reinforcement learning for mobile gui agents. *arXiv preprint arXiv:2509.18119*, 2025.

[46] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*, 2024.

[47] Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026.

[48] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[49] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

[50] Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, et al. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*, 2025.

[51] Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

[52] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[53] Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zhenyu Hou, Zehan Qi, Hanyu Lai, Yifan Xu, et al. Agentrl: Scaling agentic reinforcement learning with a multi-turn, multi-task framework. *arXiv preprint arXiv:2510.04206*, 2025.

[54] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Appendix

A Action Space and System Prompt

Our agent receives screenshot observations and interacts through each benchmark's exposed `computer` action interface. We do not use accessibility trees or privileged environment state for perception. Below we provide the complete action space and system prompts for both evaluation environments.

A.1 OSWorld Action Space

The agent receives screenshots at each turn and issues actions through the following interface:

- **Mouse actions:** `left_click`, `right_click`, `middle_click`, `double_click`, `triple_click`, `left_click_drag`, `mouse_move`, `left_mouse_down`, `left_mouse_up` — each parameterized by `(x, y)` pixel coordinates.
- **Keyboard actions:** `key` (press a key or combination, e.g., `ctrl+s`), `hold_key` (hold for a duration), `type` (type a string of text).
- **Scroll:** `scroll` with direction (`up/down/left/right`) and amount.
- **Utility:** `screenshot` (capture current screen), `wait` (pause for a duration), `cursor_position` (get current cursor location).

Coordinates use a relative 1000×1000 grid that is rescaled to the actual screen resolution (1280×800) by the environment harness.

A.2 OSWorld System Prompt

System Prompt for OSWorld

<SYSTEM_CAPABILITY>
You are utilising an Ubuntu virtual machine using x86_64 architecture with internet access. You can install Ubuntu applications with your bash tool. Use curl instead of wget. To open browser, please just click on the Chrome icon. Using bash tool you can start GUI applications, but you need to set `export DISPLAY=:1` and use a subshell. GUI apps run with bash tool will appear within your desktop environment. When viewing a page, zoom out or scroll down to see everything before deciding something isn't available. DO NOT ask users for clarification during task execution. The screen's resolution is 1000×1000. Home directory is `/home/user`. Password for sudo is `password`.
</SYSTEM_CAPABILITY>

A.3 ScienceBoard Action Space

The ScienceBoard environment extends the OSWorld action space with three additional actions for scientific workflow support:

- `done`: Signal task completion (for action-oriented tasks).
- `answer`: Submit a computed value (for measurement/computation tasks), with the answer in the `text` field.
- `write_file`: Write content to a file on the VM (file path in `text`, content in `content` field), supporting Unicode. This avoids the need to open a terminal for file operations.

All other mouse, keyboard, scroll, and utility actions are identical to OSWorld.

A.4 ScienceBoard System Prompt

System Prompt for ScienceBoard

<SYSTEM_CAPABILITY>
You are utilising an Ubuntu virtual machine with scientific software installed. The machine runs ScienceBoard applications: KAlgebra, Celestia, ChimeraX, GrassGIS, TeXstudio, and Lean/VS Code. You interact with the desktop via mouse and keyboard actions. You can see the screen through screenshots and must use GUI interaction to complete tasks.
</SYSTEM_CAPABILITY>

<APPLICATION_HINTS>
- **KAlgebra**: mathematical calculator; use `:=` for variable assignment.
- **Celestia**: space simulator; use GUI menus.
- **ChimeraX**: molecular visualization; use command line at bottom.
- **GrassGIS**: GIS tool; use GUI and terminal for GRASS commands.
- **TeXstudio**: LaTeX editor; use Ctrl+R for Find & Replace.
- **Lean/VS Code**: theorem prover in Lean 4 with Mathlib; write proofs via `write_file` action.

</APPLICATION_HINTS>

B VERIGEN Pipeline Details

This section provides the full set of VERIGEN details referenced from the main text, including an anonymized example of a generated task, the multi-agent filtering ablation, the per-domain task distribution, judge-correctness audits, capability-ordered validation on held-out tasks, the utility of trajectory-guided augmentation, and an overlap audit against benchmark evaluation pools.

B.1 Example Generated Task Schema

Table 7 shows an anonymized example of a VERIGEN-generated task. The concrete task content, identifiers, URLs, and local paths are masked, while the schema fields are preserved to illustrate how each task couples an executable setup with a task-specific Python judge.

B.2 Task Context and Trajectory-Guided Examples

Document, slide, and spreadsheet tasks use collected and formatted seed documents; media and code tasks use collected files or generated project artifacts. Email, OS, and multi-app tasks can be grounded in explored application state. Scientific tasks use specialized application environments and their supported state interfaces. In the OSWorld audit subset, 733/1,049 tasks load 1,315 setup artifacts; estimated task length averages 6.5 steps with a maximum of 35, and parsed source trajectories have a median of 15 turns. These are descriptive task/source statistics, not measured solve times. Table 8 breaks the audited subsets down by source-trajectory outcome.

Success-driven hardening (Thunderbird). A successful source trajectory exports 30 contacts from the Personal Address Book through menus, file dialogs, desktop saving, and a spreadsheet conversion. The generated variant adds a prerequisite: create a new contact, then export all contacts to `all_contacts.csv`. The verifier checks that the CSV includes the original contacts and the inserted contact/email. This changes both the required operation sequence and final artifact.

Failure-driven decomposition (Chrome and Calc). A source trajectory fails to compare unseen movies on IMDB and instead copies local movie rows to Calc, but demonstrates sheet creation, copying, and sorting. The generated task isolates those capabilities: create a `movies_90s` sheet with only 1990–1999 movies, preserve headers, and sort ratings in descending order. Verification checks the sheet, headers, row count, year constraints, and ordering. This removes the failed web-comparison subgoal while retaining a verifiable spreadsheet task.

Table 7: An anonymized VERIGEN task example. The example preserves the generated schema while masking task-specific identifiers, file names, URLs, and local paths.

Field	Anonymized Content
`id`	`<task-uuid>`
`new_id`	`<task-uuid>_<augmentation-tag>`
`snapshot`	`<desktop-application-snapshot>`
`instruction`	Apply a specified presentation-wide style change to all textual content in a desktop office document.
`source`	`<public-help-page-or-application-reference>`
`setup config`	Download `<input-file>` to the VM desktop and open it with the target application.
`related_apps`	`[<target-desktop-application>]`
`postconfig`	Activate the target application window, wait for UI stabilization, execute a save action, and wait until the file state is persisted before judging.
`judge func`	`check_<app>_<property>__<hash>`
`result getter`	Inspect the final environment state, such as document metadata, application state, accessibility-tree properties, or generated files.
`expected rule`	A task-specific rule, e.g., the final document must satisfy a specified formatting or state constraint.
`reward`	A deterministic scalar score returned by the Python judge, typically binary for all-or-nothing tasks and dense when sub-goals are available.
`metadata`	Natural-language explanation, generated getter/metric paths with masked local roots, estimated difficulty, estimated steps, augmentation type, and validation records.
`validation`	LLM-judge pass flag and score, rule-validation pass flag, execution-in-the-loop pass flag, and optional repair suggestions.

Table 8: Source-trajectory provenance in the audited subsets.

Dataset	With source trajectory	Successful source	Non-successful source
OSWorld	1,049	561	488
ScienceBoard	825	375	450
Total	1,874	936	938

Table 9: Distinct task-tree stages and audit subsets. The audited subsets are contained in the RL-task tree and must not be added to the release total.

Scope	OSWorld	ScienceBoard	Total
Candidate records	20,289	4,366	24,655
All released task files	–	–	27,867
Audited trajectory-guided subsets	1,049	825	1,874

Figure 7 shows the corresponding anonymized Python judge. The getter retrieves the final artifact from the environment, extracts task-specific state variables, and the metric converts the extracted state into a deterministic reward.

VERIGEN Judge Function (anonymized)

```
def get_document_property__<hash>(env, config):
    """Extract task-specific properties from the final desktop artifact."""
    file_bytes = env.controller.get_file(config["path"])
    if not file_bytes:
        return set()

    with tempfile.NamedTemporaryFile(suffix="<artifact-ext>",
                                     delete=False) as tmp:
        tmp.write(file_bytes)
        tmp_path = tmp.name

    try:
        document = load_desktop_artifact(tmp_path)
        observed_values = set()
        for page in document.pages:
            for element in page.elements:
                if has_relevant_content(element):
                    value = extract_target_property(element)
                    if value:
                        observed_values.add(value)
        return observed_values
    finally:
        os.unlink(tmp_path)


def check_document_property__<hash>(result, expected, **options):
    """Return a deterministic reward for the task-specific rule."""
    expected_value = expected.get("<target_property>")
    if not result:
        return 0.0
    return 1.0 if result == {expected_value} else 0.0
```

Figure 7: Anonymized Python judge corresponding to the task schema in Table 7. The concrete artifact type, function suffix, target property, and file paths are masked, while the generated evaluator structure is preserved.

B.3 Release Counts and Audit Scope

Table 9 separates these scopes. The raw candidate directories include unused candidates and controls. Their counts and distributions characterize released artifacts, not an accepted training pool. The paper-subset OSWorld manifest excludes imported-batch records; the ScienceBoard manifest excludes instruction-rewrite controls. These manifest boundaries define the overlap and semantic-audit evidence. We do not extrapolate the subset findings to all released RL-task records or to the entire training history. A complete checkpoint-to-training-manifest audit remains necessary for a global train–evaluation separation claim.

B.4 Extended Analysis of VERIGEN

Running VERIGEN at its 100+ parallel-worker scale produces over 24K candidate tasks, of which nearly 3K pass all filtering stages to form the final RL task pool. This pool is more than an order of magnitude larger than what prior desktop CUA methods produce for either supervised warm-up or RL (Figure 8, left), placing SCALECUA on a distinctly different point of the data-scale axis. The generated tasks span all 10 OSWorld application domains and all 6 ScienceBoard scientific domains (per-domain distribution and end-to-end synthesis cost in Appendix B.5). Coverage extends beyond office productivity to scientific applications, with tasks grounded in concrete artifacts and

Table 10: Target-type counts across all 27,867 released task files. Counts do not describe only accepted or audited training tasks.

Task/verifier target	Records
Spreadsheet/table content	4,552
Browser/web state and research	3,792
Presentation/slide editing	3,220
OS settings, filesystem, commands	2,099
Document formatting/content	2,057
Code/editor/terminal workflows	1,904
Image editing/media artifacts	1,574
Email/client workflows	1,268
GIS/map layer state	1,122
Astronomy simulation state	1,081
Media playback/player state	1,047
Symbolic math/plot state	1,023
Molecular visualization state	937
Lean proof compilation/checks	838
Multi-app artifact workflows	825
LaTeX file/compile checks	528
Total	27,867

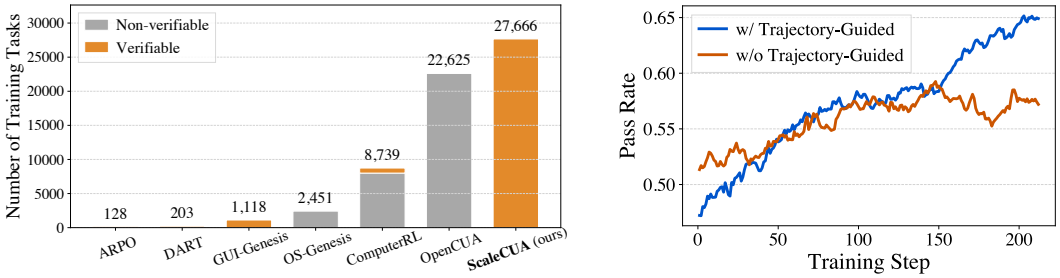

Figure 8: VERIGEN analysis. **Left:** VERIGEN produces an order-of-magnitude larger verifiable task pool than prior desktop CUA methods. **Right:** augmenting the RL pool with trajectory-guided tasks consistently raises task-level reward throughout training.

environment states; release statistics (24,655 candidate records within a 27,867-file released task tree), domain distributions, and examples are provided in Appendices B.3, B.5, and B.2.

We further evaluate the quality and training utility of the generated tasks from two angles. For judge correctness, an expert human-judge audit on 10 trajectories per application domain shows that the executable judges agree with human labels on **82.5%** of cases (details in Appendix B.7), indicating that the reward signals are not only mechanically executable but also broadly consistent with expert assessment. The same 160-example audit yields 90.8% verifier-positive precision; its 10 false positives and 18 false negatives also identify residual reward errors that executability checks alone cannot rule out. A complementary audit of 1,874 trajectory-guided tasks examines verifier semantics and train–evaluation overlap: no exact instruction duplicates are found, although some semantic overlap remains. These are subset-level findings; the semantic census and overlap analysis are reported in Appendices B.7 and B.11. For training utility, augmenting the RL pool with trajectory-guided tasks raises the OSWorld test score from 64.6% to **68.7%** (and the training-subset pass rate as shown in Figure 8, right), confirming the practical utility of VERIGEN beyond raw task count. Figure 8 (right) shows that trajectory-guided augmentation consistently improves task-level reward throughout training. This sustained separation complements the final-score ablation in Section 4.3: the generated tasks improve the learning trajectory as well as the endpoint. Recycling successful trajectories into harder tasks and failed trajectories into simpler subgoals ties task generation to demonstrated agent capabilities, providing useful training variation beyond increasing the raw pool size.

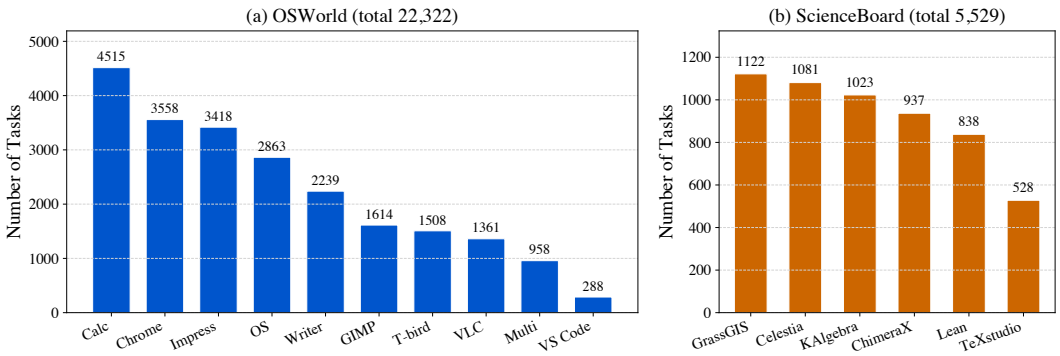

Figure 9: Per-domain task distribution of VERIGEN-generated tasks. (a) OSWorld: 10 application domains, total 22,322 tasks. (b) ScienceBoard: 6 scientific domains, total 5,529 tasks. Counts aggregate across all generation stages.

B.5 Multi-Agent Filtering Ablation and Task Distribution

Table 5 (in the main text) ablates the VERIGEN multi-agent filtering loop by removing each module. The full pipeline achieves 94.5% executable rate for generated judge functions; removing the LLM Judge Agent causes the largest drop (-32.2%), followed by the Fix Agent (-16.4%) and Rule Validator (-8.3%).

Figure 9 shows the per-domain distribution of VERIGEN-generated tasks for OSWorld (10 application domains) and ScienceBoard (6 scientific domains), aggregating across all generation stages. Coverage spans every domain, with the largest volumes in office productivity (Calc, Impress, Writer) and browsing (Chrome) on OSWorld and balanced coverage across scientific applications on ScienceBoard.

End-to-end synthesis cost. Each synthesis-worker call generates three candidates in parallel and passes through rule validation, LLM-judge validation, and executable Docker validation. Across all runs, the measured executable rate is 94.5%, with an average end-to-end latency of 221.7 seconds and 0.93–1.01 USD per accepted task. At 100+ parallel workers, this translates into a daily throughput of roughly 35K candidate tasks under steady-state operation.

B.6 Generation Roles, Settings, and Acceptance Criteria

All generation LLM roles use `claude-sonnet-4-6` through Claude Code in the reported configuration. No explicit temperature or top-p override is set by the CLI wrapper. The proposer asks for three candidates per call. Each candidate includes setup, instruction, evaluator/getter code or ScienceBoard evaluation steps, expected targets, and provenance metadata. Table 11 lists the roles and their required outputs.

The default admission logic requires successful rule/semantic validation, executable setup and evaluation, and no positive reward for states that should fail. Negative controls include no-op/wait, safe mouse movement, and wrong-answer controls where applicable. Passing these checks is not a proof of task solvability or complete semantic correctness. The optional `agent_solve` result estimates the pool's difficulty under a selected policy and is not used to accept or reject individual tasks. Consequently, there is no default rejection category for "valid but unsolved by the checker."

Prompt availability and implementation scope. The public release at `https://github.com/THUDM/SCALE-CUA` contains the inspected implementation's full prompt builders for OSWorld and ScienceBoard proposal, trajectory guidance, semantic judging, fixing, and optional action generation, together with their referenced skill documents. The proposer is file-based: it directs the coding agent to read environment and task-construction skills rather than embedding all documentation in a single string. Task paths, source trajectories, and output directories are runtime substitutions. These files document the current reproducible implementation; they are not represented as an archive of every historical rendered prompt or exact training-time repository revision.

Table 11: Generation role inputs and responsibilities. The deterministic checker is distinct from the optional action agent.

Role	Inputs	Required output or check
Proposer	Environment documentation, source task or trajectory, reusable artifacts	Three setup–instruction–verifier candidates
Semantic judge	Instruction, setup, evaluator and referenced code	Alignment assessment and repair suggestions
Rule validator	Candidate schema and generated code	Structural and syntax checks
Dynamic checker	Reset environment and candidate evaluator	Executability; reject initial-state/negative-control false positives
Fixer	Candidate, validation errors and suggestions	Repaired task/code, revalidated by the same loop
Optional action agent	Sampled task, screenshot, action contract	Pool-level solve-rate diagnostic

B.7 Expert Human-Judge Audit

To directly characterize judge correctness, we randomly sample 10 VERIGEN-generated trajectories from each application domain in OSWorld and ScienceBoard and ask expert human reviewers to inspect the trajectory evidence and final state. As shown in Table 12, executable judge decisions agree with expert human labels on 82.5% of the 160 audited examples, with 78.0% agreement on OSWorld and 90.0% on ScienceBoard. This audit complements the executable-rate and model-capability validations by providing a direct estimate of judge-human agreement and residual false positive/false negative cases.

Table 12: Expert human-judge audit of VERIGEN-generated trajectories. We randomly sample 10 examples per application domain from OSWorld and ScienceBoard. Human Valid reports the fraction judged successful by expert human reviewers. Agreement compares the executable judge decision with the human label; FP/FN count judge false positives and false negatives against the human label.

Benchmark	**Domain**	**N**	**Human Valid (%)**	**Agreement (%)**	**FP**	**FN**
OSWorld	Chrome	10	70.0	70.0	1	2
OSWorld	GIMP	10	90.0	80.0	0	2
OSWorld	LibreOffice Calc	10	90.0	90.0	1	0
OSWorld	LibreOffice Impress	10	80.0	80.0	1	1
OSWorld	LibreOffice Writer	10	80.0	70.0	2	1
OSWorld	Multi-Apps	10	70.0	90.0	0	1
OSWorld	OS	10	100.0	60.0	0	4
OSWorld	Thunderbird	10	80.0	70.0	2	1
OSWorld	VLC	10	70.0	80.0	1	1
OSWorld	VS Code	10	90.0	90.0	0	1
OSWorld	**Overall**	**100**	**82.0**	**78.0**	**8**	**14**
ScienceBoard	Celestia	10	30.0	90.0	1	0
ScienceBoard	ChimeraX	10	80.0	100.0	0	0
ScienceBoard	GrassGIS	10	40.0	100.0	0	0
ScienceBoard	KAlgebra	10	100.0	80.0	0	2
ScienceBoard	Lean	10	50.0	80.0	0	2
ScienceBoard	TeXstudio	10	50.0	90.0	1	0
ScienceBoard	**Overall**	**60**	**58.3**	**90.0**	**2**	**4**
All	**Overall**	**160**	**73.1**	**82.5**	**10**	**18**

Complete AI-assisted verifier census. The semantic census uses `claude-sonnet-5` at temperature zero on all 1,874 audited task–verifier pairs. Inputs include instruction, setup, evaluator code or structured steps, expected values, and metadata; ScienceBoard additionally uses verified runtime semantics for its six applications. PASS means fully aligned; WEAK means aligned in the primary objective but incomplete or fragile; FAIL means incorrect or unusable; UNSURE means insufficient static evidence. These are AI-assisted semantic judgments, not measured rollout error rates or independent human labels. PASS and WEAK account for 76.4% and 21.7%, respectively. OSWorld has

Table 13: Adjudicated semantic census. WEAK is not counted as PASS.

Dataset	N	PASS	WEAK	FAIL	UNSURE
OSWorld	1,049	764	270	0	15
ScienceBoard	825	667	136	2	20
Total	1,874	1,431	406	2	35

one high-false-positive-risk verifier under this audit. ScienceBoard's low/medium/high risk counts are 678/126/21; 124 of its 136 WEAK cases omit secondary requirements. The two confirmed FAIL cases are a missing ChimeraX electrostatic-potential-coloring check and two inconsistent KAlgebra surface points. These localized findings do not imply that all remaining verifiers are error-free. In particular, the older JSON-only ScienceBoard audit was superseded by runtime-informed adjudication because Lean's active proof-checking step could otherwise be misread as a vacuous placeholder.

Expert-audit correction and errors. The 160-example expert audit yields TP=99, TN=33, FP=10, and FN=18 after checking labels against the recorded human and environment decisions. Agreement remains $(99 + 33)/160 = 82.5\%$, and verifier-positive precision is $99/109 = 90.8\%$. OSWorld has 8 FP and 14 FN (78.0% agreement, 89.5% precision); ScienceBoard has 2 FP and 4 FN (90.0% agreement, 93.9% precision). Table 12 gives the per-domain breakdown and Table 14 the error taxonomy.

Table 14: Taxonomy of the 28 expert-audit disagreements.

Error	Cause	Count
FP	Partial completion accepted	4
FP	Wrong target or global state accepted	3
FP	Infeasible/missing-resource non-completion accepted	2
FP	Malformed artifact accepted	1
FN	Visible UI success missed	9
FN	Valid terminal/filesystem/system path not credited	5
FN	Correct proof/math result not credited	4

False positives can reward incorrect behavior, while false negatives under-credit useful behavior. Neither the audit nor the use of benchmark-owned evaluators quantifies the full causal impact of reward noise on learning. They separate training-reward errors from direct evaluation-score computation and identify concrete repair targets.

Scoring-form audit. ScienceBoard uses the fraction of passed evaluation steps, and many OSWorld verifiers aggregate checks over artifact properties, files, cells, or settings. The detailed scoring CSV contains 554 verifiers with one parsed component, 227 with two, 851 with three or four, 166 with five to nine, and 32 with at least ten; 44 have zero parsed components (unresolved by this parser, not necessarily zero reward checks). The earlier aggregate combined these 44 with the single-component group, giving 598. Thus 1,276/1,874 (68.1%) have at least two parsed components. Separately, 1,290/1,874 (68.8%) use explicit partial-credit/step-wise scoring and 535/1,874 (28.5%) use numeric tolerance or graded similarity. Feature flags can overlap and are not equivalent to component counts; for example, a structured multi-check flag need not imply multiple separately parsed reward components. Partial credit can make a reward less sensitive to one failed subcheck, but incomplete checks can still produce false positives.

Table 15: Validation of generated tasks across different model capabilities (150 tasks per type).

Model	Task Augmentation		Trajectory Augmentation	
	Avg. Score	Pass Rate (%)	Avg. Score	Pass Rate (%)
Qwen3-VL-8B-Base	13.0	12.6	19.0	14.0
Claude-4.5-Sonnet	55.5	52.6	59.0	50.0
Gap (Sonnet−Base)	**+40.0**		**+36.0**	

Reward-noise sensitivity status. A low-risk OSWorld pool retains 933 tasks: 764 PASS plus 169 WEAK tasks with low false-positive risk and none/minor severity. A Qwen3.5-9B pilot trained with this pool and an additional task source reaches 64.1 average clamped OSWorld reward at checkpoint step 275 (360 tasks, four repeats); its best-of-four diagnostic is 76.9. This is not a matched noise-removal ablation against 68.7: pool composition and run conditions differ, and the training record includes restarts and environment-capacity pressure. We retain it as a sensitivity diagnostic and do not infer that filtering causes the score difference or eliminates reward hacking.

B.8 Capability-Ordered Task Validation

A complementary concern is whether the generated tasks are logically solvable, meaningful, and appropriately difficult for online RL. We evaluate two models of differing capability—Qwen3-VL-8B-Base and Claude-4.5-Sonnet—on a held-out subset of 150 tasks per augmentation type (task-only and trajectory-guided). As shown in Table 15, Claude-4.5-Sonnet achieves a pass rate above 50%, showing that many VERIGEN tasks are solvable and that judge errors are not the dominant failure mode. The Base model achieves around 13% pass rate, suggesting that tasks are non-trivial for weaker models while remaining solvable by stronger agents.

B.9 Trajectory-Guided Task Augmentation

Figure 8 (right) compares RL training with and without trajectory-guided task augmentation, evaluated on the same capability-driven validation subset. Adding trajectory-guided tasks—derived from rollout experience as simpler variants of failed tasks or harder extensions of solved ones—consistently improves task-level reward throughout training, complementing the overall benchmark gain from 64.6% to 68.7% reported in the main text.

B.10 Component and Task-Source Ablation Details

The component ablation compares full SCALECUA (68.7%) with removal of Frontier Sampling (63.7%) or Visual Context Segmentation (62.2%). The 43.9% base row is our measured starting point and uses no generated pool or corresponding RL and therefore does not isolate generator quality. For task-source decomposition, the Qwen3.5-9B RL recipe and evaluation protocol are fixed: single-proposer tasks with basic filtering yield 55.4%, multi-agent validation and repair yield 64.6%, and full trajectory guidance yields 68.7%. The gains of 9.2 and 4.1 points support validation/repair and trajectory guidance beyond RL on a generated pool. Task-source construction changes across rows; the comparison is not task-count matched.

B.11 Training vs. Evaluation Overlap Audit

To check whether VERIGEN training tasks duplicate benchmark evaluation tasks, we audit the generated subsets used for training against the corresponding benchmark task pools. The audit separates provenance metadata from task-content overlap: source IDs indicate which seed environment or task inspired generation, but they are not treated as leakage unless the generated task also copies the objective or judge. We report exact full-JSON duplication, normalized-instruction duplication, objective near-duplicates using a composite instruction-similarity threshold of 0.90, and exact judge reuse.

As shown in Table 16, the training subsets contain no full-JSON duplicates and no exact instruction duplicates. Objective near-duplicates are rare (0.19% on OSWorld and 0.36% on ScienceBoard).

ScienceBoard has no exact reuse of evaluation judges; OSWorld shows a small amount of exact evaluator reuse (2.10%), corresponding to reusable generic evaluator templates rather than copied task instructions. The ScienceBoard instruction-rewrite control subset intentionally reuses source evaluators and is excluded from training.

Table 16: Training-data overlap audit against benchmark task pools. We report only the generated subsets used for training. The ScienceBoard instruction-rewrite control subset, which intentionally reuses source evaluators, is excluded from training and omitted here.

Benchmark	Generated subset	N	Full JSON duplicates	Instruction exact duplicates
OSWorld	traj_verify	1049	0.00% (0)	0.00% (0)
ScienceBoard	sci_traj_verify	825	0.00% (0)	0.00% (0)

Benchmark	Objective near-duplicates	Judge exact reuse	Objective similarity p50 / p95
OSWorld	0.19% (2)	2.10% (22)	0.319 / 0.628
ScienceBoard	0.36% (3)	0.00% (0)	0.342 / 0.596

Multi-field train–evaluation similarity. The audit compares the 1,049 OSWorld tasks with 369 official evaluation tasks and the 825 ScienceBoard tasks with 169 official tasks. Each pair is represented by instruction/objective, setup, evaluator logic and getter schema, observed/result-state specification, and expected target state. Similarity is the maximum of word-bigram and character 3–5-gram TF-IDF cosine similarities, with custom verifier hash suffixes canonicalized. Joint scores take the minimum across the required fields for the *same* evaluation task, preventing unrelated per-field neighbors from being combined. Full-task similarity requires agreement on instruction, setup,

Table 17: Number of audited tasks with a qualifying same-pair nearest neighbor.

Dataset	Fields	$\geq .80$	$\geq .85$	$\geq .90$	$\geq .95$
OSWorld	Objective	10	7	3	0
	Instruction + target	5	4	2	0
	Full task	2	2	0	0
ScienceBoard	Objective	9	5	4	0
	Instruction + target	8	0	0	0
	Full task	8	0	0	0

evaluator, and final state. No full-task pair reaches 0.90, and no objective pair reaches 0.95. However, lexical thresholds can miss paraphrases. AI adjudication with `claude-sonnet-5`, temperature zero, classifies the 19 objective candidates at 0.80 as 13 related-but-distinct, four same-objective variants, and two same-task paraphrases. All ten full-task candidates at 0.80 are related-but-distinct. The six objective-level variants/paraphrases represent 0.32% of the audited subset. The conclusion is limited benchmark-pattern overlap, not absence of overlap.

Original audit versus stricter canonicalization. Table 16 preserves the original definitions: objective-near counts of 2/1,049 and 3/825, and 22/1,049 (2.10%) OSWorld exact judge reuse. The newer multi-field analysis has different text representations and gives 3/1,049 and 4/825 objective neighbors at 0.90. A separate canonical evaluator-hash rerun gives 33 OSWorld matches: 18 infeasibility checks and 15 generic settings/template checks. These are distinct definitions, not interchangeable estimates. Generic checker reuse alone does not establish task copying, but some objective-level variants remain; in particular, Python-extension installation in VS Code warrants caution. None of these analyses certifies the full released pool or an overlap-removal training run.

C Sliding-Window Details

This section provides implementation details for Sliding-Window Trajectory Segmentation: the token-stream update algorithm, the ablation configurations used in the main-text compute sweep, and a finer-grained view of how K affects task success.

C.1 Trajectory Processing Algorithm

Token-ID stream. Policy-gradient training requires the rollout log-probabilities and training tokens to stay aligned, which can be broken if the same history is re-tokenized under different BPE boundaries. We therefore maintain the stream incrementally with *token diffs*: response tokens and their log-probabilities are appended as trainable tokens, while new observation tokens are appended with zero loss mask. After pruning, the retained history is re-tokenized with a zero loss mask, so previous assistant responses only provide context in later segments and their losses are not recomputed. Algorithm 2 gives the full procedure; all segments from the same rollout share the trajectory-level reward and advantage.

Algorithm 2 Sliding-window trajectory processing with token-ID stream

Require: Episode with T turns; K = images to keep; Δ = min removal threshold
Ensure: Set of trace segments $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$; $n_{\text{img}} \leftarrow 0$
Tokenize initial history $\mathcal{H} \rightarrow$ token-ID stream $(\mathbf{ids}, \mathbf{mask}, \mathbf{logp})$ with $\mathbf{mask}{=}\mathbf{0}$
for $t = 1, \ldots, T$ **do**
  Generate response tokens $(\mathbf{ids}_t^{\text{gen}}, \mathbf{logp}_t^{\text{gen}})$ from policy
  $\mathbf{ids} \mathrel{+}= \mathbf{ids}_t^{\text{gen}}$; $\mathbf{mask} \mathrel{+}= \mathbf{1}$; $\mathbf{logp} \mathrel{+}= \mathbf{logp}_t^{\text{gen}}$ ▷ Trainable tokens
  Execute action, receive observation with screenshot; $n_{\text{img}} \leftarrow n_{\text{img}} + 1$
  if $n_{\text{img}} > K + \Delta$ **then** ▷ Pruning triggered
    Save segment: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\mathbf{ids}, \mathbf{mask}, \mathbf{logp})\}$
    Remove oldest Δ images from $\mathcal{H}$; $n_{\text{img}} \leftarrow n_{\text{img}} - \Delta$
    Re-tokenize $\mathcal{H} \rightarrow (\mathbf{ids}, \mathbf{mask}, \mathbf{logp}) \leftarrow (\text{new_ids}, \mathbf{0}, \mathbf{0})$ ▷ Fresh stream
  else ▷ Token-diff append
    $l \leftarrow |\textsc{Tokenize}(\mathcal{H})|$ ▷ Length before appending o_t
    $\mathbf{diff} \leftarrow \textsc{Tokenize}(\mathcal{H} \oplus o_t)[l:]$ ▷ New obs tokens only
    $\mathbf{ids} \mathrel{+}= \mathbf{diff}$; $\mathbf{mask} \mathrel{+}= \mathbf{0}$; $\mathbf{logp} \mathrel{+}= \mathbf{0}$ ▷ Non-trainable tokens
  end if
end for
Save final stream: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\mathbf{ids}, \mathbf{mask}, \mathbf{logp})\}$
Assign trajectory reward r to all segments in $\mathcal{S}$
return $\mathcal{S}$

C.2 Training Ablation Configurations

Table 18 reports the rollout-to-training worker ratio used for each combination of sliding-window size K and compute scale in the ablation study (Section 4.3). For all configurations, the batch size is fixed at 16 distinct trajectory UIDs and rollout-n at 8. Smaller K values produce more segments per episode with shorter sequences, so fewer rollout workers and more training workers are allocated. Larger K values produce fewer but longer segments, requiring more rollout workers to keep the training engine fed. In practice, 4-node and 8-node setups can use larger batch sizes, which would further amplify the U-shaped trend in Figure 5(b) by increasing training-engine pressure at small K and rollout-engine pressure at large K; the relatively flat 8-node curve at large K reflects that batch size 16 underutilizes the available parallelism and partially masks the rollout-engine bottleneck.

C.3 Impact of K on Task Success

Figure 10 reports pass@k (k=1, 2, 4, 8) on OSWorld for the Qwen3.5-9B trained checkpoint under $K \in \{1, 3, 5, 8, 10, 15\}$. Across all pass@$k$ metrics, K=3–5 consistently achieves the highest scores, with pass@1 peaking at 58.9% (K=5) compared to 56.4% (K=1) and 56.8% (K=15). Too small

Table 18: Rollout-to-training node allocation for sliding-window training ablation experiments. All runs use Qwen3.5-9B [48] with batch size 16 and rollout-n=8. Rollout ratio denotes the fraction of total nodes dedicated to rollout.

K	Total Nodes	Rollout Ratio	Rollout Nodes	Training Nodes	Avg Step Time (s)
2 Nodes (16 GPUs)					
1	2	0.25	0.5	1.5	1102.6
3	2	0.25	0.5	1.5	643.8
5	2	0.50	1.0	1.0	703.4
8	2	0.50	1.0	1.0	687.5
10	2	0.50	1.0	1.0	1042.6
15	2	0.50	1.0	1.0	1649.3
4 Nodes (32 GPUs)					
1	4	0.25	1	3	962.0
3	4	0.25	1	3	525.8
5	4	0.25	1	3	405.2
8	4	0.50	2	2	400.2
10	4	0.75	3	1	612.6
15	4	0.75	3	1	693.3
8 Nodes (64 GPUs)					
1	8	0.25	2	6	502.9
3	8	0.25	2	6	279.4
5	8	0.50	4	4	302.9
8	8	0.50	4	4	282.5
10	8	0.50	4	4	434.8
15	8	0.75	6	2	519.9

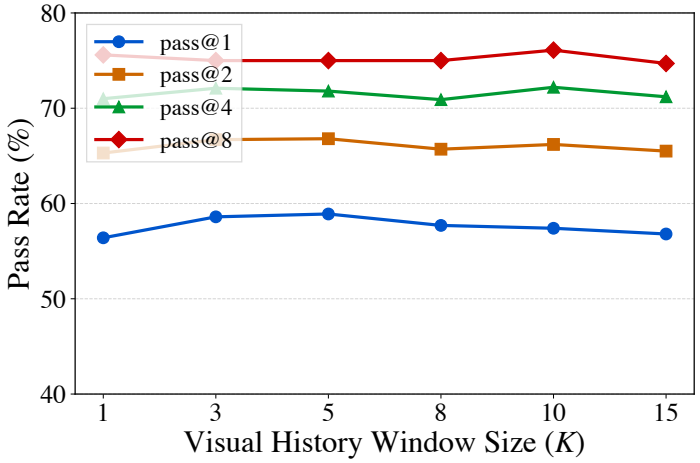

Figure 10: Pass@k on OSWorld under different sliding-window sizes K. Bounded visual context does not reduce task success, and moderate windows (K=3–5) perform best.

a window causes the agent to lose track of earlier context, while too large a window dilutes visual attention across stale screenshots.

C.4 Case Study: Impact of Visual Context Window

To qualitatively illustrate why the window size K matters beyond aggregate metrics, we examine a representative multi-app task: *"Save all open Chrome blog articles as PDFs to `/home/user/Documents/Blog`, named by article title."* This task requires persistent awareness of the global goal (target directory, title-based naming) while executing a multi-step save workflow across browser tabs. We evaluate the checkpoint (Qwen3.5-9B [48]) with $K \in \{1, 3, 5, 8, 15\}$ using

8 rollouts each. A clear inverted-U pattern emerges: K=5 achieves 7/8 pass rate, K=3 reaches 5/8, while K=1, K=8, and K=15 all score 0/8. With K=1, the agent completes the save workflow mechanically but fails to rename files in these rollouts—it loses track of the naming requirement after a few steps and saves with default Chrome filenames. With $K \geq 8$, stale visual context from early-episode actions (terminal commands, file manager dialogs) dilutes attention, causing the agent to enter repetitive action loops and inflating execution time by 2×–3×. K=5 retains exactly the right amount of recent history—enough to remember the target directory, naming convention, and remaining tabs, without interference from irrelevant earlier screenshots. This confirms that the optimal K identified by training speed analysis also maximizes task success: moderate visual context preserves essential state without overwhelming the model's attention capacity.

D Training Details

Table 19 summarizes the hyperparameters used in the RL stage.

RL stage. Online RL is conducted using AgentRL [53] with GRPO [30] as the base algorithm. We train for up to 1,000 iterations with a learning rate of 3×10^{-6} (no weight decay). Each iteration collects n=8 rollouts per task with a batch size of 32 tasks (16 for ablation studies) and a concurrency of 600 parallel VM environments. The ViT backbone is frozen during RL; only the language model parameters are updated. The rollout engine uses vLLM with TP=2, while the training engine uses Megatron with TP=4. Rollout sampling temperature is 0.8. Each episode runs for a maximum of 50 turns.

We adopt asymmetric clipping from DAPO [52]: ϵ_l=0.2, ϵ_h=0.28, with a clip ratio cap of c=3.0. A KL penalty $\lambda = 1 \times 10^{-4}$ regularizes against the reference policy. Incomplete episodes (agent fails to finish within the turn limit) receive a reward penalty of -0.2. For Frontier Sampling, tasks are ranked by Eq. (2); the top $3{\times}B$ tasks form the frontier candidate pool, $(1{-}\gamma)B$ tasks are sampled from this pool according to w_i, and γB tasks are sampled uniformly from outside the pool.

RL objective. At each iteration, Frontier Sampling selects a task batch $\mathcal{B}$ according to Eq. (2). For each task $i \in \mathcal{B}$, the rollout policy π_{rollout} generates n trajectories, yielding rewards $\{r_{i,j}\}$ from judge functions and trace segment sets $\{\mathcal{S}_{i,j}\}$ from Visual Context Segmentation. The group-relative advantage $\hat{A}_{i,j}$ is computed over the n rollouts of each task and shared across all segments of the same rollout, while loss masks ensure each assistant response is optimized exactly once:

$$\mathcal{L}(\theta) = -\mathbb{E}_{i\sim\mathcal{B}}\, \mathbb{E}_{j=1}^{n} \sum_{s\in\mathcal{S}_{i,j}} \sum_{t\in s} \Big[\min\Big(\rho_t\, \hat{A}_{i,j},\ \text{clip}(\rho_t, 1{-}\epsilon_l, 1{+}\epsilon_h)\, \hat{A}_{i,j}\Big)\Big] + \lambda\, D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})\,, \tag{3}$$

where $\rho_t = \pi_\theta(a_t \mid x_{<t}) / \pi_{\text{rollout}}(a_t \mid x_{<t})$ is the importance sampling ratio, ϵ_l and ϵ_h are the lower and upper clipping ranges, λ is the KL coefficient, and the inner sum runs over trainable tokens ($\mathbf{mask}_t$=1) within each segment.

Infrastructure and compute. The main RL experiments use 8 nodes, each equipped with 8 NVIDIA H800 (80 GB) GPUs (64 GPUs total). The rollout engine is allocated 25% of the GPU resources (16 GPUs) and the remaining GPUs are used by the actor/reference training engine. Table 20 reports the RL-stage training compute profile, and Table 21 summarizes the environment-side rollout resources. Checkpoints are saved every 25 iterations.

Table 19: RL hyperparameters.

Hyperparameter	**Value**
Algorithm	GRPO
Learning rate	3×10^{-6}
Clip ratio (low / high)	0.2 / 0.28
Clip ratio cap c	3.0
KL penalty λ	1×10^{-4}
Entropy coefficient	0.0
Batch size (tasks per iter)	32
Rollouts per task n	8
Max iterations	1,000
Rollout temperature	0.8
Max turns per episode	50
Max sequence length	40,000
Incomplete episode penalty	-0.2
Sampling hyperparameters	
Target success rate μ	0.5
Bandwidth σ	0.25
Random sample ratio γ	0.2
EMA smoothing α	0.2
Initial pass rate range $[p_{\min}, p_{\max}]$	$[0.2, 0.8]$
Sliding-window training hyperparameters	
Images to keep K (OSWorld)	5
Images to keep K (ScienceBoard)	8
Min removal threshold Δ	5

Table 20: RL-stage training compute profile. We report the standard resource allocation used for the main RL runs. Average training step time is computed from the logged `timings/step` metric of representative production runs.

Item	**OSWorld RL**	**ScienceBoard RL**
GPU cluster	8 H800 nodes, 64 GPUs total	
Rollout ratio	0.25	
Rollout GPUs	16 GPUs	
Actor/reference GPUs	48 GPUs	
Rollout engine	vLLM, TP=2	
Training engine	Megatron-LM, TP=4	
Task batch size	32	16
Rollouts per task	8	8
Environment concurrency	600	300
Adaptive sampling σ	0.25	0.15
Visual window K	5	8
Avg. training step time	672.1 s	379.0 s

Table 21: Environment-side compute profile for online GUI rollouts. Environment workers are Docker containers and are matched one-to-one with rollout workers.

Item	**Value**
Environment backend	OSWorld / ScienceBoard Docker containers
Peak Docker concurrency	600 containers
Rollout-worker mapping	1 Docker container per rollout worker
Average rollout wall-clock time	539.9 s
Environment lifecycle	Containers are deployed, used for interaction, and cleaned automatically
Action horizon	Up to 50 interaction turns per rollout

E Additional Evaluation Protocols and Results

E.1 Additional Evaluation Protocols

Table 22: OOD results of OSWorld-RL checkpoints and their bases (%). Δ: RL minus Base, in percentage points.

Benchmark	Backbone	Base	RL	Δ
ScienceBoard	Qwen3.5-9B	6.8	10.1	+3.3
	GLM-4.6V-Flash	5.8	10.2	+4.4
	Qwen3-VL-8B	3.8	17.0	+13.2
OpenApps	Qwen3.5-9B	51.8	59.8	+8.0
	GLM-4.6V-Flash	43.8	45.5	+1.7
	Qwen3-VL-8B	67.0	58.9	−8.1
OfficeWorld	Qwen3.5-9B	6.1	25.6	+19.5
	GLM-4.6V-Flash	4.5	21.9	+17.4
	Qwen3-VL-8B	7.1	24.0	+16.9

ScienceBoard probes professional scientific applications; OpenApps probes everyday application interactions across calendar, to-do, messenger, and maps tasks. Table 22 compares each OSWorld-RL checkpoint with its corresponding base without training on either target benchmark. Scores are averaged across four repeats, never maximized. The ScienceBoard results are separate from the benchmark-specific training in Table 4.

For the added average-reward analyses, with N tasks and four repeats, we use

$$\bar{r} = \frac{1}{N}\sum_{i=1}^{N} \bar{r}_i, \qquad \bar{r}_i = \frac{1}{4}\sum_{j=1}^{4} \max(0, r_{ij}). \tag{4}$$

This is distinct from best-of-four, $N^{-1}\sum_i \max_j \max(0, r_{ij})$. Clamping removes negative task-limit penalties for benchmark reporting; it does not remove penalties from RL training. Original main-table subscripts report standard deviation across evaluation passes. Model-scale pilot SEM is the sample standard deviation of task means divided by $\sqrt{N}$; repeats for one task are not counted as four independent tasks. These uncertainty measures are not interchangeable.

ScienceBoard transfer. Each of the six base/RL evaluations contains 169 tasks $\times$ four repeats = 676 records, with no malformed records or missing final reward files in the verified summaries. Unrounded base/RL average rewards are 0.067554/0.100616 for Qwen3.5, 0.057717/0.102096 for GLM, and 0.038215/0.169576 for Qwen3-VL. These use OSWorld-RL checkpoints, unlike the benchmark-specific ScienceBoard training behind Table 4.

OpenApps transfer. The study uses the 28 original tasks with screenshot-only observations, temperature zero, seed 42, 1280×720 viewport, and a 50-step limit. Four repeated runs are averaged. Infrastructure-invalid episodes are rerun before aggregation. The Qwen3.5 base and RL adapters both disable implicit thinking in this screen-only harness. Table 23 reports mean accuracy and sample standard deviation across runs. This fixed-configuration experiment is not the full OpenApps environment-variation reliability study. Navigation is near saturation. Functional-task scores show a larger spread and account for the Qwen3-VL regression. The four-repeat statistics do not quantify variation across training seeds.

E.2 OfficeWorld Evaluation

OfficeWorld [20] contains 180 real tasks; the loader's 181st entry is a non-task sentinel. All runs use four repeats and a 50-turn limit with 1280×720 images. The RL "direct-answer" runs add system guidance for formula/script/text-answer tasks (including `ofw-2-1`): answer in assistant text and avoid

Table 23: OpenApps accuracy (%), mean ± run-level standard deviation.

Model	Overall	Functional	Navigation
Qwen3.5 Base	51.8 ± 4.6	22.1 ± 5.6	97.7 ± 4.5
Qwen3.5 RL	59.8 ± 1.8	33.8 ± 2.9	100.0 ± 0.0
GLM Base	43.8 ± 1.8	7.4 ± 2.9	100.0 ± 0.0
GLM RL	45.5 ± 1.8	11.8 ± 0.0	97.7 ± 4.5
Qwen3-VL Base	67.0 ± 1.8	45.6 ± 2.9	100.0 ± 0.0
Qwen3-VL RL	58.9 ± 2.1	32.4 ± 3.4	100.0 ± 0.0

unnecessary GUI use unless live-file modification is required. Scores report mean reward $\times 100$; Table 22 gives the per-backbone values. The spreadsheet subset mixes ordinary file manipulation with tasks expecting formulas, VBA, macros, or textual answers. Some RL trajectories overuse GUI tools on text-answer tasks. For Qwen3.5, direct-answer guidance changes the `ofw-2-1` score from 2.27 to 9.09; for GLM, from 10.44 to 15.61; Qwen3-VL remains approximately 10.23. The GLM runs also contain multimodal generation failures.

E.3 Model-Scale Pilots

Table 24: OSWorld model-scale pilots: average reward (%) ± task-level SEM, with a 50-turn limit.

Category	Qwen3.5-4B	Qwen3.6-35B-A3B
OS	82.3 ± 6.3	91.7 ± 4.7
Office	68.5 ± 3.5	74.9 ± 3.4
Daily	69.1 ± 4.8	74.6 ± 4.4
Professional	83.7 ± 4.3	89.8 ± 3.9
Workflow	41.7 ± 4.4	51.5 ± 4.7
Overall	64.8 ± 2.2	72.1 ± 2.1
Available task IDs	360	359

The larger pilot uses Qwen3.6-35B-A3B [27]. The selected checkpoints are step 100 for the 4B pilot and step 75 for the 35B-A3B pilot. Office combines Calc, Writer, and Impress; Daily combines Chrome, Thunderbird, and VLC; Professional combines GIMP and VS Code. Category uncertainty is computed after grouping task IDs, not by averaging per-application SEMs. The official OSWorld set has 369 tasks: these pilots cover 360/369 and 359/369, leaving nine and ten tasks unavailable. The 4B and 35B-A3B Workflow categories contain 92/101 and 91/101 tasks, respectively. Scores average available tasks and do not impute missing outcomes. The headline pilot scores are average rewards; the 4B best-of-four diagnostic, 77.8, is not the reported average.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state three contributions—scaling verifiable task synthesis with VERIGEN, Frontier Sampling for upstream rollout allocation, and Visual Context Segmentation for efficient multi-turn training—all directly substantiated by experiments in Sections 4.2–4.3.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The *Limitations* paragraph discusses bounded interaction budgets, generalization beyond Ubuntu, and the need for systematic scaling studies under matched training and evaluation budgets.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper proposes a system framework and empirical study; it does not contain formal theoretical results requiring proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 3 fully describes the VERIGEN pipeline, Frontier Sampling, and Visual Context Segmentation; Appendix D provides hyperparameters and infrastructure configurations; Appendix B.6 documents the generation models, role inputs, and admission criteria, together with the released prompt builders; and a public code release is referenced in the abstract.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: A public code repository at `https://github.com/THUDM/SCALE-CUA` is linked from the abstract, containing the VERIGEN pipeline, training scripts, and instructions to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 describes the training framework and online RL setup; Appendix D provides RL hyperparameters (Table 19) and compute resources (Table 20); Appendix B.10 gives the ablation protocols. For the transfer results, Appendix E.1 specifies the reward aggregation, clamping, and four-repeat conventions, Appendix E.2 documents the OfficeWorld setting including the direct-answer system guidance applied to the RL runs, and Appendix E.3 reports the model-scale pilot checkpoints and task coverage.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Main-table subscripts report standard deviations across four rollout passes; scores are averaged, not maximized. Appendix E.1 distinguishes these from OpenApps run-level standard deviations and model-scale task-level SEM; the transfer summary tables report four-pass averages without error bars.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix D reports node configurations (2/4/8 nodes), 600-container concurrency, average rollout wall-clock time, and the 100+ parallel synthesis-worker setup used for VERIGEN; Table 18 gives the per-configuration rollout-to-training node allocation. The transfer evaluations and the model-scale pilots reuse the same evaluation infrastructure; Appendices E.1 and E.3 report their task counts, repeat counts, and selected checkpoints rather than separate node-hour budgets.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research uses publicly available benchmarks (OSWorld, ScienceBoard, OpenApps, OfficeWorld) and synthetic tasks generated within sandboxed Docker environments; no personally identifiable information or sensitive user data is collected, and the work conforms to the NeurIPS Code of Ethics.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: The paper focuses on methods and empirical evaluation and does not include a societal-impact analysis.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release any new pretrained large model. Our released artifacts are RL-finetuned checkpoints based on existing open-source backbones and synthetic GUI tasks confined to sandboxed Docker environments, which do not pose high misuse risk beyond what already exists in the underlying open-source models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All external assets are cited at first use with their original papers or model cards, and are used under their respective licenses: the OSWorld, ScienceBoard, OpenApps, and OfficeWorld benchmarks; the GLM-4.6V-Flash, Qwen3-VL-8B, Qwen3.5-9B, Qwen3.5-4B, and Qwen3.6-35B-A3B backbones; the AgentRL, vLLM, and Megatron-LM training and serving stacks; and the Claude models used inside VERIGEN and the audits (Appendices B.6 and B.7).

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The public release contains a README, training scripts, and configuration files documenting the VERIGEN pipeline and the SCALECUA training stages, together with the generated task tree and the role prompt builders. Appendix B.3 documents the release scope (27,867 task files, of which 24,655 are candidate records) and states explicitly that the candidate directories include unused candidates and controls and are not an accepted training pool; Table 10 gives the target-type distribution and Appendix B.1 shows the task schema.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing. The expert human-judge audit (Appendix B.7) was conducted by a small group of internal authors and does not constitute crowdsourcing or external human-subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human-subjects research. No IRB approval is required for the internal expert audit on synthetic GUI tasks.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: VERIGEN uses LLMs as core reasoning engines that drive task synthesis: they read application documentation, explore live OS environments, and generate task instructions together with executable Python judge functions. This usage is described in Section 3.2.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.