Double-Helix Co-Training for Computer-Use Generator and Verifier Models

Anonymous Authors[1]

Abstract

Reinforcement learning for computer-use agents requires accurate reward signals, but such signals are often hard to obtain at scale. Large language model (LLM) judges provide a convenient source of feedback, yet they can be biased toward fluent model-generated behavior and yield false positives. We propose a generator–verifier co-training framework, which we refer to as *Double-Helix Co-Training*. The verifier is trained using matched-state preferences that compare trusted demonstration actions, optionally including checker-filtered transitions, against the generator's counterfactual alternatives at the same history state. The generator is optimized using verifier-induced rewards, and step scores are aggregated into a conservative trajectory reward via a min or soft-min operator. We further interpret the alternating updates through a generalized EM view, which motivates trust-region regularization and clarifies the stabilizing dynamics. Empirically, our verifier achieves state-of-the-art precision on AgentRewardBench, and the co-trained 7B generator is a strong byproduct that is competitive on ScreenSpot-V2, ScreenSpot-Pro, and Mind2Web.

1. Introduction

Building agents that can use computers to do tasks such as navigating websites, operating GUIs, and writing or executing code in a sandbox requires reliable multi-step decision-making. In many realistic settings, supervision is sparse—for example, a final binary success signal, the result of unit tests, or a small set of invariants. This makes reinforcement learning from verifiable rewards (RLVR) an appealing yet challenging direction, where rewards are computed by objective checks such as tests, exact-answer matching, or rule-based validators. RLVR has gained popularity for training language models with externally checkable rewards (Su et al., 2025; Wen et al., 2025) and has recently been adapted to GUI action prediction with rule-based feedback (Lu et al., 2025). However, RLVR is only effective when the training feedback is reliable and informative at the level of individual actions; training multi-step computer-use behaviors remains difficult.

A common strategy is to rely on programmatic verifiers when available, or to learn reward signals from preference data and use them to guide policy improvement. Recent work on computer-use has also made rapid progress along two main modeling directions. First, supervised GUI grounding models localize UI elements or predict low-level actions from instructions (Hong et al., 2024; Shaw et al., 2023; Cheng et al., 2024), and reinforcement-based post-training further improves spatial accuracy with shaped rewards (Lu et al., 2025; Luo et al., 2025; Zhou et al., 2025). Second, end-to-end agents trained on large trajectory corpora combine planning and grounding in a single model or modular system (Qin et al., 2025; Xu et al., 2025; Wang et al., 2025), often augmented with synthetic data generation to scale task and behavior diversity (Lai et al., 2024; He et al., 2025; Xu et al., 2024; Pahuja et al., 2025). When programmatic checks are unavailable, many of these approaches rely on learned evaluators, including LLM judges, to provide dense feedback.

A core difficulty is that an evaluator must remain reliable on the states and actions produced by the current policy. As the policy explores new interface states and tool outputs that are weakly covered by offline supervision, a fixed evaluator can make systematic errors. Optimizing against these errors encourages shortcuts that look good to the evaluator but do not complete the task, leading to brittle improvements.

Inspired by prover-verifier games (Kirchner et al., 2024), we introduce an alternating generator–verifier co-training loop, which we refer to as *Double-Helix Co-Training*. The key mechanism is matched-state counterfactual supervision: at the same history state, the verifier is trained to prefer trusted demonstration steps over counterfactual alternatives proposed by the current generator. As a result, whenever

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

the generator finds a high-scoring failure mode, it produces exactly the hard negatives needed to correct the verifier in the next round. To turn step judgments into a single training signal, we aggregate step scores with a conservative min or soft-min operator, so a trajectory cannot score highly if any step is low quality. We also provide a generalized EM interpretation of the alternating updates, which motivates trust-region regularization and helps explain why the procedure stabilizes.

In addition, our generator departs from many prior agents that emit a fixed schema of executable GUI actions (Qin et al., 2025; Xu et al., 2025; Wang et al., 2025). Instead, we decouple planning from execution at the representation level by producing natural-language action descriptions that specify the intended interaction with the interface. This makes actions easier to evaluate and allows execution to be delegated to interchangeable grounding agents, enabling modular transfer across environments while reusing strong open-source grounders (Cheng et al., 2024; Gou et al., 2024; Wu et al., 2024a). The design is consistent with language-based reasoning and action decoupling (Yao et al., 2022; Shinn et al., 2023; Ahn et al., 2022). While the verifier is our primary deliverable, this same co-training procedure produces a competitive generator as a byproduct.

We summarize our contributions as follows: (i) We introduce an alternating generator–verifier co-training procedure for computer-use, where matched-state counterfactual preferences grounded in demonstrations continually repair the verifier as the generator improves. (ii) We develop **Verifier-7B**, a step-level generative verifier that achieves SOTA on AgentRewardBench (Lù et al., 2025). (iii) As a byproduct of co-training, we obtain a natural-language computer-use generator that is competitive on ScreenSpot-V2, ScreenSpot-Pro, and Mind2Web.

2. Preliminaries

2.1. Computer-Use as a History-MDP

Computer-use tasks are interactive and partially observed. At step t, the agent receives an observation o_t summarizing the computer state (e.g., screenshot, UI tree, tool outputs) and produces a natural-language action description u_t that specifies the intended interaction. An executor model then maps u_t into an executable computer-use action a_t (e.g., click/type/scroll/hotkey, or a tool call such as running code).

Because o_t is not Markov, we define the state as the interaction history $s_t \triangleq (o_0, u_0, \ldots, u_{t-1}, o_t)$, yielding a history-MDP for each task x. The executor produces an executable action $a_t = \textsc{Ground}(s_t, u_t)$, and the environment executes it and returns the next observation:

$$o_{t+1} \sim \textsc{Execute}(s_t, a_t), \qquad a_t = \textsc{Ground}(s_t, u_t). \tag{1}$$

The next state appends the new interaction: $s_{t+1} = (s_t, u_t, o_{t+1})$. Episodes terminate either when the environment ends the task or when $t = T - 1$.

Supervision signals. True task reward is typically unavailable. Some transitions expose programmatic checks, such as tests, invariants, or rule-based validators, denoted $v_{t+1} = \textsc{Verify}(s_t, a_t, s_{t+1})$, which can be sparse or delayed. We use such checks only for filtering or labeling when constructing training data, not as a dense reward. For a subset of tasks we also have trusted solution trajectories $\tau^\star = (s_0, a_0^\star, s_1, a_1^\star, \ldots)$.

2.2. Step-Level Preferences

From demonstrations, each visited state s_t yields a ground-truth action description $u_t^\star$. We construct step-level preference triples (s, u^+, u^-), where $u^+ = u^\star$ is the demonstrated description and u^- is a counterfactual alternative sampled from the generator policy at the same state, $u^- \sim \pi_\theta(\cdot \mid s)$.

We train a step-level verifier that assigns a scalar score $V_\phi(s, u) \in \mathbb{R}$, where larger values indicate more correct and useful steps. We model preferences at a matched state s with a Bradley-Terry form: $\Pr[u^+ \succ u^- \mid s] = \sigma\big(V_\phi(s, u^+) - V_\phi(s, u^-)\big)$. In our implementation, the verifier is realized as a generative judge conditioned on (s, u^+, u^-), and we extract $V_\phi(s, u)$ from calibrated preference likelihoods. Training details and score extraction are provided in Sec. 3.1.

3. Method

We propose a co-training framework for interactive computer-use that couples (i) a *generator* policy π_θ that emits natural-language action descriptions and (ii) a *step-level verifier* that scores step quality. Training alternates between improving the generator under verifier-induced rewards and improving the verifier on preference feedback constructed from demonstrations and generator rollouts. We formalize this training framework as a *generalized EM* procedure, which helps explain the error-correction mechanism and why the coupled dynamics stabilize.

3.1. Step-Level Verifier Model

At demonstrated states s_t we treat the demonstrated description $u_t^+ = u_t^\star$ as preferred. We draw counterfactual descriptions $u_t^- \sim \pi_\theta(\cdot \mid s_t)$, and retain tuples that can be confidently labeled using demonstrations and/or checkers. This yields a step preference dataset $\mathcal{D}_{\text{pref}} = \{(s_t, u_t^+, u_t^-)\}$.

We implement the verifier as a generative judge $p_\phi(\cdot \mid$

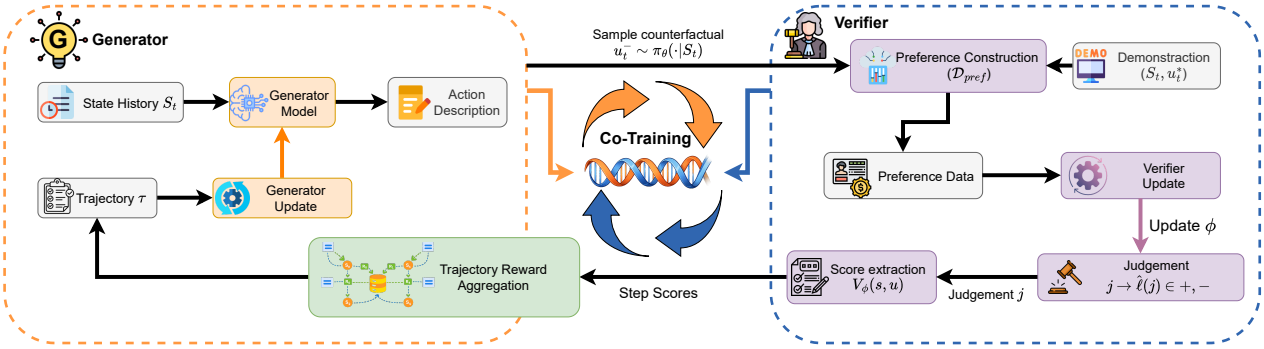

Figure 1. Overview of the training framework. The generator produces an action description from the state history, while the verifier learns from matched-state preferences between demonstrations and generator counterfactual actions. Verifier step scores are aggregated into a conservative trajectory reward to update the generator, and the loop alternates.

s, u^+, u^-) that outputs a short judgment j ending in a discrete decision $\hat{\ell}(j) \in \{+,-\}$ indicating whether u^+ or u^- is preferred. We use a label-matching reward for the judge:

$$R_{\text{ver}}(j) = \mathbb{1}\{\hat{\ell}(j) = \ell\} - \mathbb{1}\{\hat{\ell}(j) \neq \ell\},$$

where $\ell \in \{+,-\}$ is the ground-truth preference label for the pair (s, u^+, u^-). In our data construction, ℓ is derived from step-level attributes such as correctness and redundancy, treating correct and non-redundant steps as preferred. We train the judge with a KL-regularized policy objective:

$$\max_{\phi} \; \mathbb{E}_{(s,u^+,u^-)\sim\mathcal{D}_{\text{pref}}} \, \mathbb{E}_{j\sim p_\phi(\cdot|s,u^+,u^-)}\big[A_{\text{ver}}(j)\big] - \beta_{\text{ver}} \, \text{KL}\Big(p_\phi(\cdot \mid s, u^+, u^-) \,\Big\|\, p_{\text{ref}}(\cdot \mid s, u^+, u^-)\Big), \tag{2}$$

where $A_{\text{ver}}(j)$ is a group-normalized advantage computed from $R_{\text{ver}}(j)$ across sampled judgments, and p_{ref} is a frozen reference model.

We extract a scalar step score $V_\phi(s,u)$ from the judge's preference likelihoods. For a given state s and candidate description u, we compare u against a baseline alternative $\tilde{u}(s)$ (e.g., a policy sample or a fixed proposal) and define

$$V_\phi(s,u) \;\triangleq\; \log \frac{\Pr_\phi[u \succ \tilde{u}(s) \mid s]}{1 - \Pr_\phi[u \succ \tilde{u}(s) \mid s]}, \tag{3}$$

where $\Pr_\phi[u \succ \tilde{u}(s) \mid s]$ is obtained by marginalizing over the final decision $\hat{\ell}(j)$ under $j \sim p_\phi(\cdot \mid s, u, \tilde{u}(s))$. In experiments, we set $\tilde{u}(s)$ to a fixed baseline proposal (e.g., the greedy decode from $\pi_{\text{ref}}(\cdot \mid s)$). This definition induces a Bradley–Terry preference model by construction.

3.2. Verifier-Induced Trajectory Reward

The generator learns from a trajectory-level scalar reward derived from step scores. For a rollout $\tau = (s_0, u_0, \ldots, s_{T-1}, u_{T-1}, s_T)$, we define

$$R_\phi(\tau) \;=\; \min_{t<T} V_\phi(s_t, u_t). \tag{4}$$

This worst-step aggregation enforces robustness: any catastrophic step collapses the trajectory reward. When helpful for optimization stability, we use a soft-min approximation:

$$R_\phi^{(\alpha)}(\tau) = -\alpha \log \sum_{t<T} \exp\left(-\frac{1}{\alpha} V_\phi(s_t, u_t)\right), \qquad \lim_{\alpha\to 0} R_\phi^{(\alpha)}(\tau) = \min_{t<T} V_\phi(s_t, u_t). \tag{5}$$

3.3. Generator Update

Fixing ϕ, we roll out trajectories under π_θ and compute $R_\phi(\tau)$. Let $\hat{A}_t$ be an advantage estimator derived from this trajectory reward (e.g., using a learned baseline). We update θ with the following:

$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\; \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big] - \beta_{\text{pol}} \, \text{KL}\Big(\pi_\theta(\cdot \mid s_t) \,\Big\|\, \pi_{\text{ref}}(\cdot \mid s_t)\Big), \tag{6}$$

where $r_t(\theta) = \pi_\theta(u_t \mid s_t)/\pi_{\theta_{\text{old}}}(u_t \mid s_t)$ and π_{ref} is a fixed reference policy.

3.4. Alternating Co-Training Algorithm

Training consists of offline initialization followed by online rounds.

Offline initialization. We (i) initialize the generator π_{θ_0} with supervised fine-tuning on demonstrations and (ii) pre-train the verifier on an offline preference set formed by pairing demonstrated steps with counterfactual sampled steps at the same states.

Online rounds. For rounds $i = 1, \ldots, R$:

1. **Generator step:** Roll out $\tau \sim \pi_{\theta_{i-1}}$, compute $R_{\phi_{i-1}}(\tau)$ via Eq. (4) (or Eq. (5)), and update θ_i by maximizing Eq. (6).

2. **Verifier step:** Construct a preference buffer $\mathcal{B}_i$ by sampling negatives $u_t^- \sim \pi_{\theta_i}(\cdot \mid s_t)$ at demonstrated states s_t, pairing with $u_t^+ = u_t^\star$, and optionally filtering using programmatic checkers. Update ϕ_i by optimizing Eq. (2) on $\mathcal{B}_i$ (optionally with replay from earlier buffers).

3.5. EM View of Generator-Verifier Training

We formalize the coupled updates as a generalized EM procedure over a latent *optimality* variable $O \in \{0, 1\}$. Let $p_\theta(\tau \mid x)$ be the trajectory distribution induced by π_θ and environment dynamics. We define an optimality likelihood via the verifier-induced trajectory reward:

$$p_\phi(O = 1 \mid x, \tau) \;\propto\; f\big(R_\phi(\tau)\big), \qquad f \text{ non-decreasing.} \tag{7}$$

Here the (omitted) normalization constant may depend on x but not on τ. We use $f(r) = \exp(r/\alpha)$ in experiments.

The marginal likelihood of optimality is

$$\log p_{\theta,\phi}(O = 1 \mid x) = \log \sum_\tau p_\theta(\tau \mid x)\, p_\phi(O = 1 \mid x, \tau).$$

Introducing a variational distribution $q(\tau \mid x)$ yields the ELBO:

$$\begin{aligned} \log p_{\theta,\phi}(O{=}1 \mid x) \;\geq\; & \mathbb{E}_{\tau\sim q(\cdot|x)}\big[\log p_\phi(O{=}1 \mid x, \tau)\big] \\ & - \mathrm{KL}\big(q(\tau \mid x) \,\|\, p_\theta(\tau \mid x)\big) \\ & := \mathcal{L}(\theta, \phi, q). \end{aligned} \tag{8}$$

E-step (reweighting / selection). Fix (θ_i, ϕ_i) and set

$$q_{i+1}(\tau \mid x) \;\propto\; p_{\theta_i}(\tau \mid x)\, f\big(R_{\phi_i}(\tau)\big), \tag{9}$$

implemented by sampling trajectories from π_{θ_i} and scoring/filtering (or importance-weighting) them using $R_{\phi_i}(\tau)$.

M-steps (two intertwined strands). Fix q_{i+1}. The generator M-step maximizes $\mathbb{E}_{q_{i+1}}[\log p_\theta(\tau \mid x)]$ (with trust-region regularization), which we approximate with PPO (Eq. (6)). The verifier M-step increases $\mathbb{E}_{q_{i+1}}[\log p_\phi(O = 1 \mid x, \tau)]$ subject to preference constraints grounded in demonstrations/checkers, which we approximate with GRPO (Eq. (2)). These alternating M-steps constitute the "double helix": policy and verifier alternately improve on the same reweighted trajectory distribution.

Scope of the EM view. We use EM as a motivating lens, not a formal guarantee: PPO/GRPO are trust-region approximations to the M-step and the refreshed-verifier schedule (Sec. 4.4) discards continuity in ϕ, so Eq. (14) does not apply literally. Under refresh, the benefit of co-training operates primarily through the *evolving negative-sampling distribution*—each round's generator supplies fresh hard negatives that reshape the preference dataset.

4. Evaluation

We evaluate (i) end-to-end task performance of the generator, (ii) reward precision of the verifier model, and (iii) round-over-round training dynamics of the coupled "double helix" loop.

4.1. Experiment Setup

Dataset and splits. We train and evaluate on the AgentNet dataset. We partition the dataset into four disjoint subsets: 10% for supervised fine-tuning (SFT), 10% for validation, 40% for generator (policy) training, and 40% for verifier training. Within the generator-training split and verifier-training split, we further divide each into three disjoint folds for the three training rounds (30%, 30%, and 40% of the corresponding split for rounds 1 to 3, respectively). This ensures that each round uses fresh tasks for data collection and updates.

Iterative training. We start by SFT the base model Qwen2.5-VL-7B on our split dataset to initialize both the generator and the verifier. We then train the generator and verifier iteratively; in our settings, we found that three rounds of training are sufficient. In each round, we (i) update the generator with Eq. 6 using the current verifier as the reward provider, and (ii) update the verifier with Eq. 2 using preference-style supervision constructed from the round's data (details in Section 3.4).

Context construction under sequence limits. AgentNet provides an image observation at every step. To stay within the model's context length, we represent the observation history using a mixed format: we include the image for the current step I_t in full, and replace previous-step images $\{I_{<t}\}$ with compact natural-language summaries of the corresponding observations and actions. Thus, the input at step t contains (a) the current image and structured state, and (b) a textual description of the past interaction history.

Output format. The generator is trained to produce, at each step, a short natural-language rationale and an action in natural language. ScreenSpot and Mind2Web require executable GUI actions (e.g., click coordinates). We therefore evaluate by running a fixed executor that converts the generator's action description into an executable action. The executor and prompts are held constant across all generator checkpoints, so reported improvements reflect better action descriptions rather than changes in the executor.

The verifier receives the same step context as the generator. For each candidate step, it predicts two step-level properties: (1) **correctness**, indicating whether the step is appropriate for progressing toward task completion, and (2) **redundancy**, indicating whether the step is unnecessary or duplicative given the past history. We convert these attributes into a single scalar step score $V_\phi(s, u)$ used for policy optimiza-

tion by treating "correct and non-redundant" as preferred in matched-state comparisons. Concretely, $V_\phi(s, u)$ is extracted from the verifier's preference likelihoods (Eq. 3), where preference supervision is constructed to penalize incorrect or redundant actions.

4.2. Verifier Model Performance

We first evaluate the trained verifier model on AgentRewardBench (Lù et al., 2025), a benchmark designed to assess the reliability of automatic evaluators for web agent trajectories using expert human annotations as ground truth. AgentRewardBench contains trajectories collected from multiple LLM-based agents across diverse web environments and evaluates whether a trajectory successfully completes the task, incurs unintended side effects, or exhibits repetitive behavior.

Following the benchmark's evaluation protocol, we report *precision* as the primary metric, since in common downstream uses of automatic evaluators—e.g., filtering trajectories for rejection fine-tuning or serving as reward signals in RL—*false positives* (failed trajectories incorrectly judged as successful) are particularly harmful and can inject substantial noise into training signals (Lù et al., 2025).

We additionally break down performance across the five included task suites: AssistantBench (AB), which measures real-world web navigation starting from a search engine; VisualWebArena (VWA), which emphasizes visually grounded web tasks; WebArena (WA), which covers diverse self-hosted web environments with text-based goals; WorkArena (Work), which targets enterprise workflows on ServiceNow; and WorkArena++ (Wk++), which increases difficulty via more compositional, multi-step enterprise tasks (Lù et al., 2025).

To align with the benchmark's trajectory-level evaluation setting (i.e., assigning a single reward to an entire trajectory), we convert each trajectory into step-level instances and label them using our verifier model. We then aggregate these step-wise scores into a final trajectory reward using Eq. 4, enabling a fair comparison to trajectory-level judges while preserving our verifier's step-level granularity.

Table 1 summarizes judge performance on AgentRewardBench, where precision is the primary metric (following the benchmark's motivation that false positives are particularly harmful when filtering trajectories for downstream training). Our Verifier-7B achieves the best overall precision among learned judges (**72.5**) with a strong F1 (73.6), outperforming all prior LLM-judge baselines. The rule-based evaluator attains higher precision (83.8) but substantially lower recall (55.9 vs. 74.9) and F1 (67.1 vs. 73.6), making our verifier a more balanced signal for downstream training. Notably, compared to the strongest in-house baseline RM-R1, Verifier-7B improves overall precision by +15.0 points (72.5 vs. 57.5) and F1 by +21.0 points (73.6 vs. 52.6), indicating substantially fewer failed trajectories are incorrectly accepted as successful. This advantage is consistent across benchmarks: Verifier-7B yields large gains on visually and instruction-sensitive settings such as VWA (68.5 vs. 49.6) and Wk++ (70.5 vs. 58.1), while also maintaining high precision on enterprise-oriented tasks (Work: 94.7 vs. 50.0). Interestingly, RM-R1 performs relatively well on AB (70.5) but degrades sharply on Work, suggesting limited robustness to workflow-heavy environments; in contrast, Verifier-7B remains strong across domains, consistent with our goal of reducing false positives via step-level, matched-state supervision and conservative worst-step aggregation.

4.3. Generator Model Performance

In this section, we evaluate how the generator model performs under co-training. We report results on ScreenSpot-V2, ScreenSpot-Pro, and Mind2Web. ScreenSpot is a single-step GUI grounding benchmark that measures whether a model can localize the correct UI element on a screenshot given a natural-language instruction, spanning mobile, desktop, and web interfaces (Cheng et al., 2024). ScreenSpot-V2 is a corrected variant that re-annotates erroneous ScreenSpot examples to reduce evaluation noise while keeping the same task format (Wu et al., 2024b). ScreenSpot-Pro further stresses fine-grained grounding in professional settings, using authentic high-resolution screens from real workflows where targets are smaller and layouts are denser (Li et al., 2025). Complementing these UI grounding benchmarks, Mind2Web evaluates web navigation by providing real-world website tasks paired with action trajectories, emphasizing generalization across diverse sites and domains (Deng et al., 2023).

Two-stage action generation Our generator is trained to produce natural-language action descriptions, not executable GUI actions such as pixel coordinates. For grounding benchmarks, emitting coordinates directly is a weak interface for verification. Coordinates are not semantically interpretable on their own, so the verifier cannot reliably judge whether an action is correct or whether it advances the task. We therefore have the generator output an explicit action description that specifies the intended interaction with the UI, which provides a meaningful signal for reward assignment.

To generate GUI actions, we adopt the key idea from Aguvis (Xu et al., 2025) of separating high-level decision-making from low-level execution. Specifically, we implement a two-stage inference pipeline. In Stage 1, the generator produces a short reasoning trace and an action description that is scored by the verifier. In Stage 2, we use the same baseline model as the generator to translate the action description into

Category	Judge	Overall			AB	VWA	WA	Work	Wk++
		Precision	Recall	F1	Precision				
Official	*Rule-based*	83.8	55.9	67.1	25.0	85.2	79.0	100.0	83.3
Existing	AER-C	67.7	71.9	69.7	83.3	56.0	68.8	100.0	66.7
	AER-V	67.6	71.5	69.5	83.3	61.2	67.6	96.4	59.3
	NNetNav-70B	52.5	82.4	64.1	20.8	54.5	54.3	77.3	43.2
Accessibility Tree	Claude 3.7 S.	68.8	81.6	74.7	87.5	61.0	69.3	85.0	66.7
	GPT-4o	69.8	83.1	75.9	77.8	63.0	**70.2**	94.6	63.0
	GPT-4o Mini	61.5	86.1	71.7	80.0	57.9	63.5	84.2	49.4
	Llama 3.3-70B	67.7	79.0	72.9	75.0	59.6	68.2	94.3	62.7
	Qwen2.5-VL	64.3	**89.8**	75.0	72.7	59.3	63.6	87.2	60.3
Screenshot	Claude 3.7 S.	69.4	76.3	72.7	71.4	64.8	69.3	85.3	66.7
	GPT-4o	68.1	80.3	73.7	77.8	60.7	69.9	93.8	59.6
	GPT-4o Mini	64.5	78.3	70.8	80.0	57.4	66.9	90.3	54.8
	Qwen2.5-VL-72B	64.5	86.1	73.7	70.0	58.5	62.9	93.8	64.4
	RM-R1-7B	57.5	48.1	52.6	70.5	49.6	61.5	50.0	58.1
Ours (Screenshot)	**Verifier-7B**	**72.5**	74.9	73.6	**89.1**	**68.5**	65.9	**94.7**	**70.5**

Table 1. AgentRewardBench performance for various judges. We report recall and F1 as auxiliary scores.

Models	Mobile		Desktop		Web		Avg
	T	I	T	I	T	I	
Generator-7B (Ours, co-trained)	98.5	81.2	92.8	69.1	92.9	78.1	87.1
Generator-7B (Ours, baseline)	97.7	81.7	92.8	67.5	90.4	76.4	86.0
SeeClick-9.6B (Cheng et al., 2024)	78.4	50.7	70.1	29.3	55.2	32.5	55.1
UGround-7B (Gou et al., 2024)	75.1	84.5	85.1	61.4	84.6	71.9	76.3
ShowUI-2B (Lin et al., 2025)	92.1	75.4	78.9	78.9	84.2	61.1	77.3
Aguvis-7B (Xu et al., 2025)	89.3	68.7	80.6	67.9	89.3	70.0	80.5
Phi-Ground-7B-16C-DPO (Zhang et al., 2025a)	96.5	62.0	90.2	76.4	93.6	75.9	83.8
OS-Atlas-7B (Wu et al., 2024a)	95.2	75.8	90.7	63.6	90.6	77.3	84.1
UI-R1-E-3B (Lu et al., 2025)	98.2	83.9	94.8	75.0	93.2	83.7	89.5
LPO (Tang et al., 2025c)	97.9	82.9	95.9	86.4	95.6	84.2	90.5
UI-TARS-7B (Qin et al., 2025)	96.9	89.1	95.4	85.0	93.6	85.2	91.6
JEDI-7B (Xie et al., 2025)	96.9	87.2	95.9	87.9	94.4	84.2	91.7
GUI-Actor-7B (Wu et al., 2025)	97.6	88.2	96.9	85.7	93.2	86.7	92.1

Table 2. Performance comparison on ScreenSpot-V2 dataset. "T" means Text, and "I" means Icon/Widget.

the concrete GUI action required by the benchmark, such as a click location, typing content, or a scroll operation. This decomposition preserves verifiability while still producing executable actions for evaluation.

Performance on ScreenSpot-V2 and ScreenSpot-Pro Tables 2 and 3 summarize results on ScreenSpot-V2 and ScreenSpot-Pro. Co-training improves our 7B generator over the baseline on both benchmarks. On ScreenSpot-V2, Generator-7B increases the overall average from 86.0 to 87.1. On ScreenSpot-Pro, which is substantially more challenging due to high-resolution professional software and smaller target elements, Generator-7B improves over the baseline from 44.6 to 45.1 average accuracy. The consistent gains across both benchmarks suggest that training under verifier-induced rewards can improve grounding performance without increasing model scale.

Performance on Mind2Web Table 4 reports performance on multimodal Mind2Web under three generalization regimes. Co-training improves both grounding and end-to-end step correctness across settings. Generator-7B increases Cross-Task element accuracy from 50.0 to 53.8 and step success rate from 41.9 to 45.6, and yields similar gains under Cross-Website and Cross-Domain evaluation. These improvements indicate that the learned generator produces more accurate element selections while maintaining robust operation prediction, which directly translates into higher step success.

4.4. Ablation Studies

In this section, we study two design choices: (i) why we train the verifier by fine-tuning the original SFT model, and (ii) why we train our model for three rounds.

Model	CAD		Dev		Creative		Scientific		Office		OS		Avg.		
	T	I	T	I	T	I	T	I	T	I	T	I	T	I	Avg
Focus-2B (Tang et al., 2025b)	7.6	3.1	22.8	1.7	23.7	1.7	25.0	7.1	23.2	7.7	17.8	2.5	19.8	3.9	13.3
UGround-7B (Gou et al., 2024)	14.2	1.6	26.6	2.1	27.3	2.8	31.9	2.7	31.6	11.3	17.8	0.0	25.0	2.8	16.5
OS-Atlas-7B (Wu et al., 2024a)	12.2	4.7	33.1	1.4	28.8	2.8	37.5	7.3	33.9	5.7	27.1	4.5	28.1	4.0	18.9
Qwen2.5-VL-7B (Bai et al., 2025)	16.8	1.6	46.8	4.1	35.9	7.7	49.3	7.3	52.5	20.8	37.4	6.7	38.9	7.1	26.8
UGround-V1-7B (Gou et al., 2024)	15.8	1.2	51.9	2.8	47.5	9.7	57.6	14.5	60.5	13.2	38.3	7.9	45.2	8.1	31.1
UI-R1-E-3B (Lu et al., 2025)	37.1	12.5	46.1	6.9	41.9	4.2	56.9	21.8	65.0	26.4	32.7	10.1	-	-	33.5
UI-TARS-7B (Qin et al., 2025)	20.8	9.4	58.4	12.4	50.0	9.1	63.9	31.8	63.3	20.8	30.8	16.9	47.8	16.2	35.7
InfGUI-R1-3B (Liu et al., 2025)	33.0	14.1	51.3	12.4	44.9	7.0	58.8	20.0	65.5	28.3	43.9	12.4	49.1	14.1	35.7
GUI-G1-3B (Zhou et al., 2025)	39.6	9.4	50.7	10.3	36.6	11.9	61.8	30.0	67.2	32.1	23.5	10.6	49.5	16.8	37.1
UI-TARS-72B (Qin et al., 2025)	18.8	12.5	62.9	17.2	57.1	15.4	64.6	20.9	63.3	26.4	42.1	15.7	50.9	17.6	38.1
JEDI-7B (Xie et al., 2025)	38.0	14.1	42.9	11.0	50.0	11.9	72.9	25.5	75.1	47.2	33.6	16.9	52.6	18.2	39.5
Phi-Ground-7B-16C-DPO (Zhang et al., 2025a)	26.9	17.2	70.8	16.7	56.6	13.3	58.0	29.1	76.4	44.0	55.1	25.8	56.4	21.8	43.2
Generator-7B (Ours)	37.1	18.8	61.0	20.0	51.5	25.9	72.2	29.1	72.9	35.9	52.3	29.2	57.1	25.7	45.1
SE-GUI-7B (Yuan et al., 2025)	51.3	42.2	68.2	19.3	57.6	9.1	75.0	28.2	78.5	43.4	49.5	25.8	63.5	21.0	47.3
GUI-G^2-7B (Tang et al., 2025a)	55.8	12.5	68.8	17.2	57.1	15.4	77.1	24.5	74.0	32.7	57.9	21.3	64.7	19.6	47.5
Qwen2.5-VL-32B (Bai et al., 2025)	74.0	21.4	61.1	13.3	38.1	15.6	78.5	29.1	76.3	37.7	55.1	27.0	63.2	22.5	47.6
Qwen2.5-VL-72B (Bai et al., 2025)	54.8	15.6	65.6	16.6	63.1	19.6	78.5	34.5	79.1	47.2	66.4	29.2	67.3	25.0	51.2

Table 3. Performance comparison on ScreenSpot-Pro dataset. "T" means Text, and "I" means Icon/Widget.

Obs.	Planner	Grounder	Cross-Task			Cross-Website			Cross-Domain		
			Ele.Acc	Op.F1	Step SR	Ele.Acc	Op.F1	Step SR	Ele.Acc	Op.F1	Step SR
T	GPT-3.5	Choice	19.4	59.2	16.8	14.9	56.5	14.1	25.2	57.9	24.1
	GPT-4	Choice	40.8	63.1	32.3	30.2	61.0	27.0	35.4	61.9	29.7
T + I	GPT-4	Choice	46.4	73.4	40.2	38.0	67.8	32.4	42.4	69.3	36.8
	GPT-4	SoM	29.6	–	20.3	20.1	–	13.9	27.0	–	23.7
I	GPT-4o	SeeClick	32.1	–	–	33.1	–	–	33.5	–	–
	GPT-4V	OmniParser	42.4	87.6	39.4	41.0	84.8	36.5	45.5	85.7	42.0
	GPT-4o	UGround	47.7	–	–	46.0	–	–	46.6	–	–
I	SeeClick-9.6B		28.3	87.0	25.5	21.4	80.6	16.4	23.2	84.8	20.8
	AGUVIS-7B		64.2	89.8	60.4	60.7	88.1	54.6	60.4	89.2	56.6
I	**Generator-7B (Ours)**		53.8	84.8	45.6	47.6	74.9	38.9	54.8	85.4	47.6

Table 4. Performance comparison on Multimodal Mind2Web across different settings. We report element accuracy (Ele.Acc), Operation F1 (Op.F1), and step success rate (Step SR). "T" means the textual HTML code as inputs. "I" means the GUI images as inputs.

As shown in Figure 2, we observe a clear degradation when the verifier is *continuously* trained across rounds, whereas *refreshing* the verifier by re-initializing from the same original SFT checkpoint at each round yields steadily improved performance. This phenomenon is also reported by Singh et al. (2023). We attribute the degradation to distribution shift: the negatives in round i are drawn from π_{θ_i}, whose occupancy drifts round-to-round, so a continuously updated ϕ chases a moving target under a low-entropy binary objective and migrates off the SFT manifold. Refreshing re-anchors ϕ and lets the verifier benefit from evolving negatives *through the data* rather than accumulated parameter drift; this is also why we treat the EM view in Sec. 3.5 as motivation rather than a formal guarantee.

In contrast, the generator is trained on a substantially harder objective that requires reasoning about the state and producing semantically grounded action descriptions. This higher-complexity prediction task appears less susceptible to rapid overfitting. As shown in Figure 3, additional rounds of generator training consistently improve performance on ScreenSpot-V2 and Mind2Web, while performance on ScreenSpot-Pro increases up to Round 2 and then declines in Round 3. Since ScreenSpot-Pro represents the most fine-grained, this trend suggests that prolonged training begins to trade off robustness on high-resolution professional UIs for gains on the other benchmarks. Based on these results, we train the generator for three rounds and use the Round 3 checkpoint in subsequent experiments.

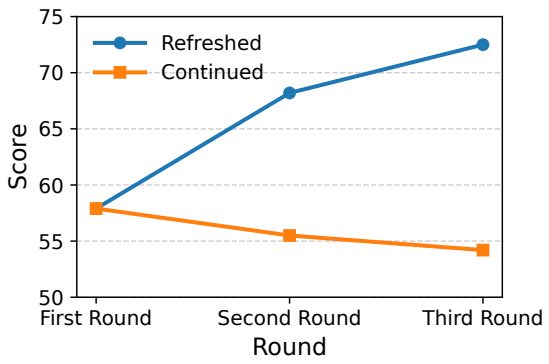

Figure 2. Continuing to train the same verifier model (*Continued*) vs. fine-tuning the original model (*Refreshed*).

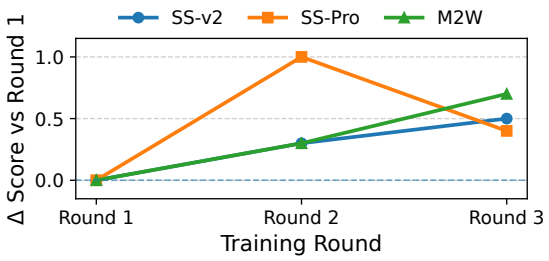

Figure 3. Generator performance trends across training rounds on ScreenSpot-V2 (SS-V2), ScreenSpot-Pro (SS-Pro), and Mind2Web (M2W). The y-axis shows the absolute change in score relative to Round 1 (Δ Score), with Round 1 normalized to 0 for each benchmark. Positive values indicate performance improvements over successive training rounds, enabling direct comparison of relative trends despite differing absolute score scales.

5. Related Work

Grounding-focused approaches aim to localize UI elements or predict low-level actions from natural language instructions, typically using supervised fine-tuning on annotated UI datasets (Hong et al., 2024; Shaw et al., 2023; Cheng et al., 2024; Gou et al., 2024; Wu et al., 2024a). To reduce annotation cost and improve robustness, recent methods adopt reinforcement-learning-based post-training with binary or continuous spatial rewards, including UI-R1 (Lu et al., 2025), GUI-R1 (Luo et al., 2025), InfiGUI-R1 (Liu et al., 2025), GUI-G1/G^2 (Zhou et al., 2025; Tang et al., 2025a), SE-GUI (Yuan et al., 2025), and LPO (Tang et al., 2025c). While effective for UI grounding, these methods typically operate at the level of individual executable actions, assuming pre-decomposed instructions and optimizing local spatial correctness, without learning to generate task-level plans or instruction decompositions.

Planning, reasoning + grounding approaches To incorporate planning, a second line of work trains agents to jointly reason and act using large-scale trajectory data. Vision-based end-to-end agents such as UI-TARS (Qin et al., 2025), AGUVIS (Xu et al., 2025), OpenCUA (Wang et al., 2025) and AgentCPM-GUI (Zhang et al., 2025b), as well as modular or multi-agent frameworks (Tan et al., 2024; Wang et al., 2024), demonstrate strong performance across GUI and web benchmarks. However, despite differences in architecture and data collection, these approaches predominantly rely on supervised fine-tuning and train planners to emit actions from a restricted, predefined action space, even in two-stage designs that separate planning from grounding or attempt to unify actions via tools such as PyAutoGUI (Xu et al., 2025; Wang et al., 2025).

We decouple planning from execution by training a computer-use generator obtained via reward-model optimization to produce high-level natural language instructions rather than executable GUI actions. Execution is delegated to interchangeable grounding agents, avoiding a fixed action schema and enabling modular, transferable computer-use. This aligns with prior language-based reasoning approaches (Yao et al., 2022; Shinn et al., 2023; Ahn et al., 2022).

Reward models and automatic evaluators for computer-use agents. Beyond sparse rule-based success signals, recent work studies learned reward models and automatic evaluators to score agent trajectories and guide training or test-time search. AgentRewardBench systematically audits LLM-based judges for web-agent trajectories and highlights that judge reliability varies substantially across environments, motivating stricter supervision and evaluator robustness (Lù et al., 2025). In parallel, RM-R1 frames reward modeling as a *reasoning* problem—generating interpretable rubrics or traces before producing preferences—which improves reward-model quality on standard RM benchmarks and provides a stronger general-purpose judge baseline (Chen et al., 2025). Our step-level verifier training aligns with this line of work, but focuses on *matched-state* counterfactual supervision and worst-step aggregation to reduce false positives under policy-induced nonstationarity.

6. Conclusion

We propose a generator-verifier co-training framework for computer-use. The generator is optimized using verifier-induced rewards aggregated by a conservative min or soft-min operator, so any low-quality step dominates the trajectory signal. The verifier is trained with matched-state preferences that contrast trusted demonstration steps, and optionally checker-filtered transitions, against counterfactual generator alternatives, turning policy improvement into targeted counterexample mining for verifier repair. Experiments show that our verifier achieves strong precision on AgentRewardBench, and that co-training improves a 7B generator on ScreenSpot-V2, ScreenSpot-Pro, and Mind2Web.

7. Impact Statement

This work studies training signals for computer-use agents by co-training a step-level verifier together with a generator. The primary intended impact is methodological: improving the reliability of automatic evaluators that can be used to score and filter trajectories or provide learning feedback. A secondary outcome is a generator that can produce natural-language action descriptions, which may support more modular agent designs.

If extended beyond benchmarks, computer-use automation could increase productivity by helping with repetitive digital tasks. At the same time, such systems could be misused to automate unwanted online behavior. Practical deployments should therefore include appropriate access controls, logging, and monitoring, and should be evaluated in the target environment before broad release.

Finally, improving data efficiency and reducing reliance on large-scale human annotation may lower training costs and energy usage for future systems.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Chen, X., Li, G., Wang, Z., Jin, B., Qian, C., Wang, Y., Wang, H., Zhang, Y., Zhang, D., Zhang, T., Tong, H., and Ji, H. Rm-r1: Reward modeling as reasoning, 2025.

Cheng, K., Sun, Q., Chu, Y., Xu, F., YanTao, L., Zhang, J., and Wu, Z. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: towards a generalist agent for the web. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 28091–28114, 2023.

Gou, B., Wang, R., Zheng, B., Xie, Y., Chang, C., Shu, Y., Sun, H., and Su, Y. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

He, H., Yao, W., Ma, K., Yu, W., Zhang, H., Fang, T., Lan, Z., and Yu, D. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27545–27564, 2025.

Hong, W., Wang, W., Lv, Q., Xu, J., Yu, W., Ji, J., Wang, Y., Wang, Z., Dong, Y., Ding, M., et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.

Kirchner, J. H., Chen, Y., Edwards, H., Leike, J., McAleese, N., and Burda, Y. Prover-verifier games improve legibility of llm outputs. *arXiv preprint arXiv:2407.13692*, 2024.

Lai, H., Liu, X., Iong, I. L., Yao, S., Chen, Y., Shen, P., Yu, H., Zhang, H., Zhang, X., Dong, Y., et al. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, 2024.

Li, K., Meng, Z., Lin, H., Luo, Z., Tian, Y., Ma, J., Huang, Z., and Chua, T.-S. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 8778–8786, 2025.

Lin, K. Q., Li, L., Gao, D., Yang, Z., Wu, S., Bai, Z., Lei, S. W., Wang, L., and Shou, M. Z. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 19498–19508, 2025.

Liu, Y., Li, P., Xie, C., Hu, X., Han, X., Zhang, S., Yang, H., and Wu, F. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*, 2025.

Lù, X. H., Kazemnejad, A., Meade, N., Patel, A., Shin, D., Zambrano, A., Stańczak, K., Shaw, P., Pal, C. J., and Reddy, S. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025.

Lu, Z., Chai, Y., Guo, Y., Yin, X., Liu, L., Wang, H., Xiao, H., Ren, S., Xiong, G., and Li, H. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 2025.

Luo, R., Wang, L., He, W., Chen, L., Li, J., and Xia, X. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

Pahuja, V., Lu, Y., Rosset, C., Gou, B., Mitra, A., Whitehead, S., Su, Y., and Hassan, A. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6300–6323, 2025.

Qin, Y., Ye, Y., Fang, J., Wang, H., Liang, S., Tian, S., Zhang, J., Li, J., Li, Y., Huang, S., et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Shaw, P., Joshi, M., Cohan, J., Berant, J., Pasupat, P., Hu, H., Khandelwal, U., Lee, K., and Toutanova, K. N. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. *Advances in Neural Information Processing Systems*, 36:34354–34370, 2023.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Singh, A., Co-Reyes, J. D., Agarwal, R., Anand, A., Patil, P., Garcia, X., Liu, P. J., Harrison, J., Lee, J., Xu, K., et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2023.

Su, Y., Yu, D., Song, L., Li, J., Mi, H., Tu, Z., Zhang, M., and Yu, D. Crossing the reward bridge: Expanding rl with verifiable rewards across diverse domains. *arXiv preprint arXiv:2503.23829*, 2025.

Tan, W., Zhang, W., Xu, X., Xia, H., Ding, Z., Li, B., Zhou, B., Yue, J., Jiang, J., Li, Y., et al. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*, 2024.

Tang, F., Gu, Z., Lu, Z., Liu, X., Shen, S., Meng, C., Wang, W., Zhang, W., Shen, Y., Lu, W., et al. Gui-g ˆ 2: Gaussian reward modeling for gui grounding. *arXiv preprint arXiv:2507.15846*, 2025a.

Tang, F., Shen, Y., Zhang, H., Chen, S., Hou, G., Zhang, W., Zhang, W., Song, K., Lu, W., and Zhuang, Y. Think twice, click once: Enhancing gui grounding via fast and slow systems. *arXiv preprint arXiv:2503.06470*, 2025b.

Tang, J., Xia, Y., Wu, Y.-F., Hu, Y., Chen, Y., Chen, Q.-G., Xu, X., Wu, X., Lu, H., Ma, Y., et al. Lpo: Towards accurate gui agent interaction via location preference optimization. *arXiv preprint arXiv:2506.09373*, 2025c.

Wang, J., Xu, H., Jia, H., Zhang, X., Yan, M., Shen, W., Zhang, J., Huang, F., and Sang, J. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *Advances in Neural Information Processing Systems*, 37:2686–2710, 2024.

Wang, X., Wang, B., Lu, D., Yang, J., Xie, T., Wang, J., Deng, J., Guo, X., Xu, Y., Wu, C. H., et al. Opencua: Open foundations for computer-use agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Wen, X., Liu, Z., Zheng, S., Ye, S., Wu, Z., Wang, Y., Xu, Z., Liang, X., Li, J., Miao, Z., Bian, J., and Yang, M. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *arXiv preprint arXiv:2506.14245*, 2025.

Wu, Q., Cheng, K., Yang, R., Zhang, C., Yang, J., Jiang, H., Mu, J., Peng, B., Qiao, B., Tan, R., et al. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*, 2025.

Wu, Z., Wu, Z., Xu, F., Wang, Y., Sun, Q., Jia, C., Cheng, K., Ding, Z., Chen, L., Liang, P. P., et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024a.

Wu, Z., Wu, Z., Xu, F., Wang, Y., Sun, Q., Jia, C., Cheng, K., Ding, Z., Chen, L., Liang, P. P., et al. Os-atlas: Foundation action model for generalist gui agents. In *The Thirteenth International Conference on Learning Representations*, 2024b.

Xie, T., Deng, J., Li, X., Yang, J., Wu, H., Chen, J., Hu, W., Wang, X., Xu, Y., Wang, Z., et al. Scaling computer-use grounding via user interface decomposition and synthesis. *arXiv preprint arXiv:2505.13227*, 2025.

Xu, Y., Lu, D., Shen, Z., Wang, J., Wang, Z., Mao, Y., Xiong, C., and Yu, T. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*, 2024.

Xu, Y., Wang, Z., Wang, J., Lu, D., Xie, T., Saha, A., Sahoo, D., Yu, T., and Xiong, C. Aguvis: Unified pure vision agents for autonomous gui interaction. In *Forty-second International Conference on Machine Learning*, 2025.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Yuan, X., Zhang, J., Li, K., Cai, Z., Yao, L., Chen, J., Wang, E., Hou, Q., Chen, J., Jiang, P.-T., et al. Enhancing visual grounding for gui agents via self-evolutionary reinforcement learning. *arXiv preprint arXiv:2505.12370*, 2025.

Zhang, M., Xu, Z., Zhu, J., Dai, Q., Qiu, K., Yang, Y., Luo, C., Chen, T., Wagle, J., Franklin, T., et al. Phi-ground tech report: Advancing perception in gui grounding. *arXiv preprint arXiv:2507.23779*, 2025a.

Zhang, Z., Lu, Y., Fu, Y., Huo, Y., Yang, S., Wu, Y., Si, H., Cong, X., Chen, H., Lin, Y., et al. Agentcpm-gui: Building mobile-use agents with reinforcement fine-tuning. *arXiv preprint arXiv:2506.01391*, 2025b.

Zhou, Y., Dai, S., Wang, S., Zhou, K., Jia, Q., and Xu, J. Gui-g1: Understanding r1-zero-like training for visual grounding in gui agents. *arXiv preprint arXiv:2505.15810*, 2025.

A. EM Derivation and Additional Convergence Details

This appendix provides (i) the ELBO derivation, (ii) the exact-EM monotonicity argument, and (iii) a generalized-EM discussion for our PPO/GRPO approximations.

A.1. ELBO Derivation

Fix a task x and consider

$$p_{\theta,\phi}(O = 1 \mid x) = \sum_{\tau} p_\theta(\tau \mid x)\, p_\phi(O = 1 \mid x, \tau).$$

Let $q(\tau \mid x)$ be any distribution over trajectories with support contained in $p_\theta(\tau \mid x)$. Then

$$\log p_{\theta,\phi}(O = 1 \mid x) \tag{10}$$

$$= \log \sum_{\tau} q(\tau \mid x)\, \frac{p_\theta(\tau \mid x)\, p_\phi(O = 1 \mid x, \tau)}{q(\tau \mid x)} \tag{11}$$

$$\geq \sum_{\tau} q(\tau \mid x)\, \log \frac{p_\theta(\tau \mid x)\, p_\phi(O = 1 \mid x, \tau)}{q(\tau \mid x)} \quad \text{(Jensen)} \tag{12}$$

$$= \mathbb{E}_{q(\cdot \mid x)}[\log p_\phi(O = 1 \mid x, \tau)] - \mathrm{KL}(q(\tau \mid x) \,\|\, p_\theta(\tau \mid x)), \tag{13}$$

which is Eq. (8).

A.2. Exact EM: E-step and M-step

Define the ELBO

$$\mathcal{L}(\theta, \phi, q) = \mathbb{E}_q[\log p_\phi(O = 1 \mid x, \tau)] - \mathrm{KL}(q \,\|\, p_\theta).$$

For fixed (θ_i, ϕ_i), maximizing $\mathcal{L}(\theta_i, \phi_i, q)$ over q yields

$$q_{i+1}(\tau \mid x) \propto p_{\theta_i}(\tau \mid x)\, p_{\phi_i}(O = 1 \mid x, \tau),$$

which is Eq. (9).

For fixed (ϕ_i, q_{i+1}), the generator M-step maximizes

$$\theta_{i+1} = \arg\max_{\theta} \mathbb{E}_{q_{i+1}}[\log p_\theta(\tau \mid x)],$$

equivalently minimizing $\mathrm{KL}(q_{i+1} \| p_\theta)$. For fixed (θ_{i+1}, q_{i+1}), the verifier M-step maximizes

$$\phi_{i+1} = \arg\max_{\phi} \mathbb{E}_{q_{i+1}}[\log p_\phi(O = 1 \mid x, \tau)].$$

In our method, this verifier objective is implemented indirectly via preference grounding; see Sec. 3.1.

A.3. Monotonicity of Exact EM (Proof Sketch)

Let q_{i+1} be the maximizer of $\mathcal{L}(\theta_i, \phi_i, q)$ over q, then

$$\mathcal{L}(\theta_i, \phi_i, q_i) \leq \mathcal{L}(\theta_i, \phi_i, q_{i+1}).$$

Let $(\theta_{i+1}, \phi_{i+1})$ maximize $\mathcal{L}(\theta, \phi, q_{i+1})$ over (θ, ϕ), then

$$\mathcal{L}(\theta_i, \phi_i, q_{i+1}) \leq \mathcal{L}(\theta_{i+1}, \phi_{i+1}, q_{i+1}).$$

Combining yields Eq. (14). If $\mathcal{L}$ is bounded above and mild regularity conditions hold, the sequence of ELBO values converges, and EM iterates converge to a stationary point of the marginal likelihood objective.

A.4. Generalized EM with PPO/GRPO

Our implementation deviates from exact EM in two places: (i) we approximate the E-step with finite rollouts and optional filtering/importance weights; (ii) we approximate M-steps with trust-region policy optimization (PPO/GRPO) rather than exact maximization.

Why trust regions help. The ELBO contains a KL term $-\mathrm{KL}(q \| p_\theta)$ that penalizes large divergence between the reweighted distribution q and the model distribution p_θ. PPO's clipping/KL regularization and GRPO's KL-to-reference regularization are practical surrogates that limit per-iteration drift, mitigating oscillations due to noise in q and nonstationarity in V_ϕ.

Preference grounding prevents reward-model drift. Unconstrained optimization of $p_\phi(O = 1 \mid x, \tau)$ can lead to self-reinforcing reward hacking. We prevent this by grounding ϕ in step-level preference constraints derived from demonstrations/checkers at matched states. As a result, exploit trajectories that contradict grounded preferences become *anti-evidence* for optimality and are suppressed.

A.5. Why alternating co-training stabilizes

Idealized monotonic improvement. In exact EM (exact E-step and exact maximization in each M-step), the ELBO in Eq. (8) is non-decreasing across iterations:

$$\mathcal{L}(\theta_i, \phi_i, q_i) \;\leq\; \mathcal{L}(\theta_i, \phi_i, q_{i+1}) \;\leq\; \mathcal{L}(\theta_{i+1}, \phi_{i+1}, q_{i+1}), \tag{14}$$

suggesting convergence to a stationary point of the EM objective under standard regularity conditions. Our algorithm is a *generalized* EM approximation: PPO/GRPO make trust-region ascent steps that empirically stabilize the updates (Appendix A).

Correction of reward misspecification. If the verifier assigns spuriously high scores to a bad region $\mathcal{R}$, then Eq. (9) increases mass on trajectories exploiting $\mathcal{R}$ because $q \propto p_\theta f(R_\phi)$. These exploit steps appear frequently as negatives u^- in the preference buffer (paired with demonstrated u^+ at the same state). The verifier update then decreases $V_\phi(s, u^-)$ relative to $V_\phi(s, u^+)$, reducing $R_\phi(\tau)$ for exploit trajectories and removing the incentive to exploit. Thus, systematic "bad-but-high-reward" regions are unstable: exploiting them generates the supervision that corrects them.

At a stable fixed point, V_ϕ is preference-consistent on the generator's occupancy distribution, and π_θ is locally optimal under the induced reward.