From Off-Policy to On-Policy: Enhancing GUI Agents via Bi-level Expert-to-Policy Assimilation

Zezhou Wang[1‡] **Ziyun Zhang**[2] **Xiaoyi Zhang**[3†]
Zhuzhong Qian[1] **Yan Lu**[3]

[1]Nanjing University [2]Peking University
[3]Microsoft Research Asia

Abstract

Vision-language models are increasingly deployed as computer-use agents (CUAs) that operate desktops and browsers. Top-performing CUAs are framework-based systems that decompose planning and execution, while end-to-end screenshot-to-action policies are easier to deploy but lag behind on benchmarks such as OSWorld-Verified. GUI datasets like OSWorld pose two bottlenecks: they expose only a few hundred interactive, verifiable tasks and environments, and expert trajectories must be gathered by interacting with these environments, making such data hard to scale. We therefore ask how reinforcement learning from verifiable rewards (RLVR) can best exploit a small pool of exist expert trajectories to train end-to-end policies. Naïvely mixing these off-policy traces into on-policy RLVR is brittle: even after format conversion, expert trajectories exhibit structural mismatch and distribution shift from the learner. We propose **BEPA** (**Bi**-Level **E**xpert-to-**P**olicy **A**ssimilation), which turns static expert traces into policy-aligned guidance via self-rolled reachable trajectories under the base policy (LEVEL-1) and a per-task, dynamically updated cache used in RLVR (LEVEL-2). On OSWorld-Verified, BEPA improves UITARS1.5-7B success from 22.87% to 32.13% and raises a held-out split from 5.74% to 10.30%, with consistent gains on MMBench-GUI and Online-Mind2Web. Our code and data are available at: https://github.com/LEON-gittech/Verl_GUI.git.

1 Introduction

GUI agents aim to solve realistic computer-use tasks—from web navigation to full desktop control—under multimodal, long-horizon interactions (Wang et al., 2024; Nguyen et al., 2024; Zhang et al., 2024; Liu et al., 2018; Zhou et al., 2024; Xie et al., 2024; Deng et al., 2023b). Recent progress largely follows two paradigms. **Framework-based** systems wrap an LLM with a compositional stack of planners, tools, and executors (Agashe et al., 2025; Xie et al., 2025), multi-agent collaboration (Ye et al., 2025), and explicit tool/code execution modules (Song et al., 2025). These frameworks can be highly effective: on OSWorld-Verified, framework-based agents such as Agent S2 and Jedi-7B w/ o3 achieve 33.00–42.40% success, whereas specialized end-to-end (E2E) policies such as UITARS1.5-7B and OpenCUA-7B are around 23–24% Table 7. In contrast, **E2E** agents train a single policy that maps screenshots and instructions directly to low-level actions (Wang et al., 2025b,a). UI-TARS (Qin et al., 2025) develops a native E2E GUI agent via multi-stage post-training on perception, unified action modeling/grounding, and reasoning, and OpenCUA (Wang et al., 2025b) releases an open framework and supervised pipeline that converts human demonstrations into state–action pairs with reflective chain-of-thought for training end-to-end computer-use agents. Despite this progress, E2E policies still lag behind strong framework-based systems on challenging benchmarks such as OSWorld-Verified (Xie et al., 2024).

Unlike text-only instruction following, where synthetic data can be scaled via Self-Instruct-style generation, GUI datasets and benchmarks such as OSWorld-Verified pose two bottlenecks: they expose only a few hundred interactive, verifiable tasks and environments, and expert trajectories must be gathered by interacting with these environments, typically by running strong framework-based agents and verifying each episode. This makes both tasks and demonstrations hard to scale and yields only a modest pool of high-quality expert traces. This raises a complementary question: *given such a small but valuable pool of expert*

[‡]: Work is done during internship at Microsoft Research Asia. [†]: Project lead.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 23581–23623

framework trajectories, how can Reinforcement Learning with Verifiable Rewards (RLVR) best exploit them to train end-to-end policies?

A natural direction is to use expert executions as off-policy guidance in RLVR. However, directly mixing such traces into learning is brittle. The core difficulty is twofold: a **structural mismatch** and a **distribution gap**. Structurally, framework traces interleave multiple roles (planning, execution, and grounding) and often operate in tool-level action spaces (e.g., APIs) that a single end-to-end policy cannot directly imitate. Distributionally, even after format conversion, the resulting trajectories can lie far from the base-policy manifold (Figure 2). In RLVR, where optimization relies on on-policy rollouts and trust-region style updates (Schulman et al., 2015), this mismatch can cause exploration collapse or unstable optimization. Mixed on-/off-policy training has been explored (Xiong et al., 2024; Yan et al., 2025; Zhang et al., 2025); LUFFY (Yan et al., 2025) instantiates this with mixed-policy GRPO (DeepSeek-AI et al., 2025) and policy shaping on *raw* off-policy traces, while BREAD (Zhang et al., 2025) introduces expert anchors and branched rollouts that let the student self-roll from expert prefixes to alleviate sparse rewards and distribution shift in textual reasoning. Yet these approaches still treat expert trajectories as largely static guidance defined in the expert's own action space; Empirically, such static integration strategies do not close the gap on OSWorld-Verified and can even degrade UITARS1.5-7B compared to pure GRPO Table 1.

We propose **BEPA** (**Bi**-Level **E**xpert-to-**P**olicy **A**ssimilation), a plug-and-play RLVR component that turns static, mismatched expert traces into dynamic, policy-aligned guidance (Figure 1 and Section 4). BEPA operates in two stages: **LEVEL-1** re-rolls expert plans under the base policy to produce *reachable* trajectories that seed an off-policy cache, and **LEVEL-2** continuously refreshes this cache using the agent's own emerging successes, injecting cached guidance only upon total exploration failure. Integrated with GRPO in Section 4.3, this design lets the agent learn primarily from on-policy exploration while using expert traces as a self-aligned scaffold. The contributions are as follows:

- We identify structural and distributional expert-to-policy mismatches in GUI agents: expert trajectories are not directly learnable by the base policy, and naive conversion remains distributionally biased (Figure 2 and Section 3).
- We propose **BEPA**, a bi-level assimilation framework that bootstraps with policy-reachable guidance and maintains dynamic alignment via a self-updating cache integrated into GRPO (Figure 1 and Section 4).
- We demonstrate the effectiveness of BEPA on three benchmarks. On OSWorld-Verified, BEPA improves UITARS1.5-7B from 22.87% to 32.13% overall success and from 5.74% to 10.30% on a strictly held-out split, while achieving consistent improvements on MMBench-GUI and Online-Mind2Web Tables 1, 8, 9 and 11. We further provide mechanism and sensitivity analyses linking performance to distribution alignment and optimization dynamics (Section 5.4, Appendix I, and Figures 3 and 5).

2 Related Work

2.1 GUI Agents

GUI agents aim to solve realistic computer-use tasks—from mobile use, web navigation to full desktop control—under multimodal, long-horizon settings (Liu et al., 2018; Zhou et al., 2024; Xie et al., 2024; Deng et al., 2023b). In this paper, we mainly focus on full desktop control scenario. Existing systems largely fall into two paradigms: **framework-based agents** and **end-to-end agents**.

Framework-based agents. Framework-based agents wrap a LLM with structured control, including hierarchical planner–executor designs (Agashe et al., 2025; Xie et al., 2025), multi-agent collaboration (Ye et al., 2025), and explicit tool/code execution modules (Song et al., 2025). Such systems can generate high-quality, executable trajectories, but their outputs are typically produced by multiple specialized roles and interfaces, making them mismatched to a single end-to-end policy.

End-to-end agents. End-to-end GUI agents instead train one policy to map observations and instructions directly to low-level actions (Wang et al., 2025b; Qin et al., 2025; Wang et al., 2025a; Gao et al., 2024; Gou et al., 2025). UI-TARS (Qin et al., 2025) develops a native end-to-end GUI agent via multi-stage post-training on perception, unified action modeling/grounding, and reasoning. ARPO (Lu et al., 2025) performs end-to-end reinforcement learning with a replay buffer to reuse success-

ful experiences across training iterations. OpenCUA (Wang et al., 2025b) instead focuses on the data and system stack for E2E CUAs, providing an open-source framework with annotation tooling and a supervised pipeline that turns recorded human computer-use demonstrations into training data for screenshot-to-action policies. Despite progress, current E2E agents remain far behind the strongest framework-based systems. This substantial gap—in contrast to the marginal improvements often seen in text-only reasoning benchmarks—makes it meaningful to study how to learn from off-policy expert traces under severe distribution shift, which is precisely the regime targeted by our expert-to-policy assimilation.

2.2 Reinforcement Learning for LLM Agents

Reinforcement learning is widely used to improve LLM agents beyond supervised imitation and preference optimization (Ziegler et al., 2019; Rafailov et al., 2023; Ouyang et al., 2022; Hu et al., 2025). Classical RLHF often relies on PPO (Schulman et al., 2017), while recent work proposes critic-free, group-based objectives such as GRPO (Shao et al., 2024; DeepSeek-AI et al., 2025) and its variants (Ahmadian et al., 2024; Liu et al., 2025; Yu et al., 2025; Chen et al., 2025). Beyond purely on-policy RL, there is growing interest in incorporating stronger experts or offline data: iterative and hybrid preference learning under KL constraints is studied in (Xiong et al., 2024); LUFFY (Yan et al., 2025) augments GRPO with off-policy teacher rollouts and a mixed-policy objective; and BREAD (Zhang et al., 2025) uses expert anchors and branched rollouts, letting the student self-roll from short expert prefixes to ease sparse rewards and distribution shift in textual reasoning. These methods show that expert-guided RL can be effective when teacher and student operate in a similar action space. In GUI agents, however, the expert action spaces and trajectory distributions could differ substantially from the base policy. BEPA re-rolls expert guidance into policy-reachable trajectories and maintains a dynamically aligned cache within RLVR, enabling stable expert assimilation under such structural and distributional mismatch.

3 Preliminaries

Following the standard end-to-end GUI agent formulation (Qin et al., 2025; Wang et al., 2025a), we consider a multi-step decision-making setting where an agent interacts with a GUI environment to complete a task specified by a natural-language instruction x. At each step t, the agent observes a screenshot $s_t \in \mathcal{S}$ and produces a *textual action trace* $a_t \in \mathcal{V}^n$, generated autoregressively by the policy $\pi_\theta(a_t \mid s_t, x)$. Unlike framework-based agents that separate high-level reasoning from low-level grounding, end-to-end GUI agents emit the *entire* reasoning and grounding sequence as a unified textual output. The action space details are provided in Appendix F.1. The environment executes a_t and transitions to s_{t+1}. Crucially, we assume access to a deterministic **verifier** function $R(\tau) \in \{0, 1\}$ that evaluates the correctness of the final system state. Given an instruction x, an episode trajectory is defined as $\tau = (x, (s_t, a_t)_{t=1}^{T})$. The sparse binary reward is typically computed at termination: $r_T = R(\tau)$, with $r_t = 0$ for $t < T$.

4 Bi-Level Expert-to-Policy Assimilation

We propose **Bi-Level Expert-to-Policy Assimilation** (BEPA), a framework designed to catalyze capability breakthroughs in end-to-end GUI agents by bridging the distributional gap between external experts and the policy's intrinsic manifold (as shown in Figure 2). Rather than passively mixing mismatched data, BEPA utilizes off-policy traces as a structured *guidance scaffold* to steer the agent beyond its initial capability frontier. As shown in Figure 1, the framework operates in two complementary stages: **LEVEL-1: Self-Rolled Execution** transforms alien expert traces into policy-compatible trajectories to initialize a guidance pool; **LEVEL-2: Self-Aligned Off-Policy Assimilation** dynamically maintains a per-task cache, injecting these guided trajectories into GRPO updates only upon *total on-policy failure* (i.e., when all rollouts in a group fail), thereby turning static expert data into an evolving, policy-aligned guidance signal.

4.1 LEVEL-1: Self-Rolled Execution

Directly training on distributionally mismatched expert data often leads to performance degradation due to significant covariate shift. To mitigate this, LEVEL-1 acts as a *reachability (policy-manifold) adapter*: it re-executes expert solutions *under the base policy* with plan conditioning, producing trajectories that are immediately executable and learnable by π_θ.

We start from an offline expert trace set $\mathcal{D}_E = \{(x, \tau_x^E)\}$, where τ_x^E is a successful trajectory

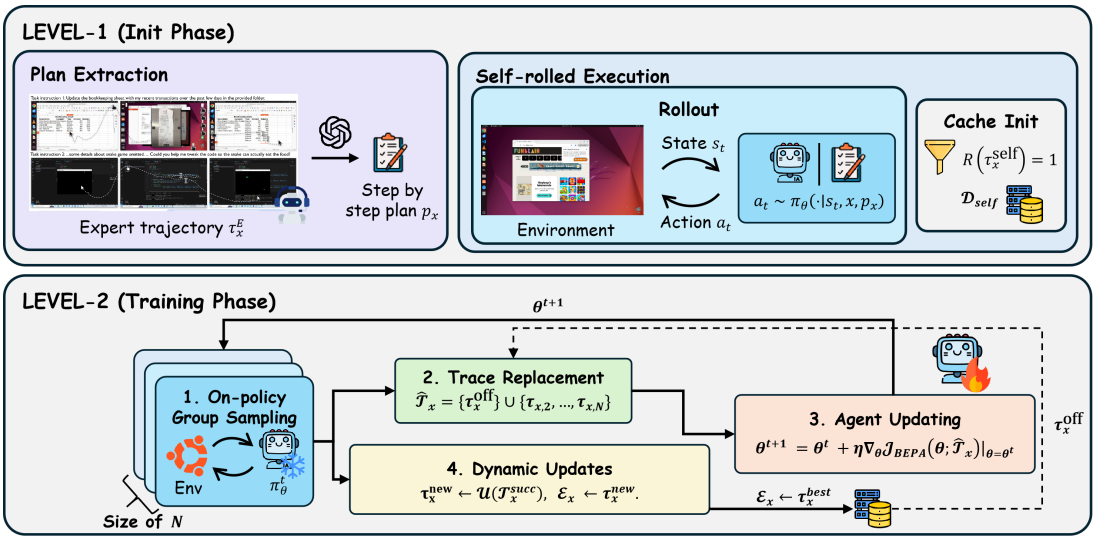


Figure 1: **BEPA overview.** We exploit strong but mismatched experts via two modular, plug-and-play stages. **LEVEL-1** initializes a policy-compatible guidance seed by re-rolling expert plans under the base policy. **LEVEL-2** maintains a self-aligned per-task cache using the agent's own emerging successes, keeping the off-policy guidance within a controllable distribution gap relative to the evolving on-policy manifold. The cached guidance is injected into GRPO only upon total on-policy failure (i.e., when all rollouts in a group fail).

from a framework-based agent or a stronger policy. For each pair (x, τ_x^E), we first abstract the expert trajectory into a compact natural-language plan $p_x = \phi(\tau_x^E)$ (e.g., a sequence of subgoals). During the *initialization phase*, we append p_x to the instruction x and let the base policy π_θ act in the environment: $a_t \sim \pi_\theta(\cdot \mid s_t, x, p_x)$. The environment executes a_t, producing a *self-rolled* trajectory τ_x^{self}. Trajectories that pass the OSWorld verifier ($R(\tau_x^{\text{self}}) = 1$) form the filtered self-rolled set:

$$\mathcal{D}_{\text{self}} = \{(x, \tau_x^{\text{self}}) : R(\tau_x^{\text{self}}) = 1\}. \quad (1)$$

The expert-derived plan p_x serves as a scaffold, guiding π_θ to visit high-reward regions it would rarely explore autonomously. Crucially, since τ_x^{self} is generated by π_θ itself, it lies much closer to the policy's manifold than the original τ_x^E.

4.2 LEVEL-2: Off-Policy Assimilation

While LEVEL-1 converts expert guidance into policy-reachable trajectories, the policy itself continues to evolve during RLVR. LEVEL-2 performs *self-aligned off-policy assimilation* by continually refreshing the cache with the policy's own successes traces, ensuring the off-policy signal evolves alongside the agent.

Self-Aligned Off-Policy Cache. We initialize $\mathcal{E}$ from the LEVEL-1 seed set $\mathcal{D}_{\text{self}}$ by setting

$$\mathcal{E}_x \triangleq \tau \quad \text{for } (x, \tau) \in \mathcal{D}_{\text{self}}, \quad (2)$$

and denote the cached trajectory by $\tau_x^{\text{off}} \triangleq \mathcal{E}_x$.

Dynamic Updates. During online GRPO training, at each iteration k, we collect a group of N on-policy rollouts $\mathcal{T}_x = \{\tau_{x,i}\}_{i=1}^N$ from $\pi_{\theta_{\text{old}}}$. Let $\mathcal{T}_x^{\text{succ}} = \{\tau \in \mathcal{T}_x \mid R(\tau) = 1\}$ denote the subset of successful trajectories. If the current policy succeeds on the task ($\mathcal{T}_x^{\text{succ}} \neq \varnothing$), we update the cache using an updating rule $\mathcal{U}(\mathcal{T}_x^{\text{succ}})$, setting $\tau_x^{\text{new}} \leftarrow \mathcal{U}(\mathcal{T}_x^{\text{succ}})$ and $\mathcal{E}_x \leftarrow \tau_x^{\text{new}}$. $\mathcal{U}(\cdot)$ is random sampling by default. In summary, LEVEL-2 keeps the guidance signal τ_x^{off} concentrated on traces that lie in high-density regions of the evolving policy π_θ, thereby reducing covariate shift during trace replacement.

4.3 Integration with GRPO

We integrate the self-aligned cache into GRPO via conditional *trace replacement*, allowing the agent to learn primarily from its own exploration and injecting off-policy guidance only upon total failure. For mixing, cached traces are injected under the original instruction x *without* the step-by-step plan prefix used in LEVEL-1 self-rolling, matching the conditioning of on-policy rollouts.

At each training step, we sample a group of on-policy rollouts $\mathcal{T}_x$ (size N) from $\pi_{\theta_{\text{old}}}$ and evaluate them using the verifier $R(\cdot)$. When the agent experiences *total exploration failure* (i.e., $\forall \tau \in \mathcal{T}_x, R(\tau) = 0$) and $\mathcal{E}_x \neq \varnothing$, we replace the first failed trajectory with the cached off-policy trajectory τ_x^{off}:

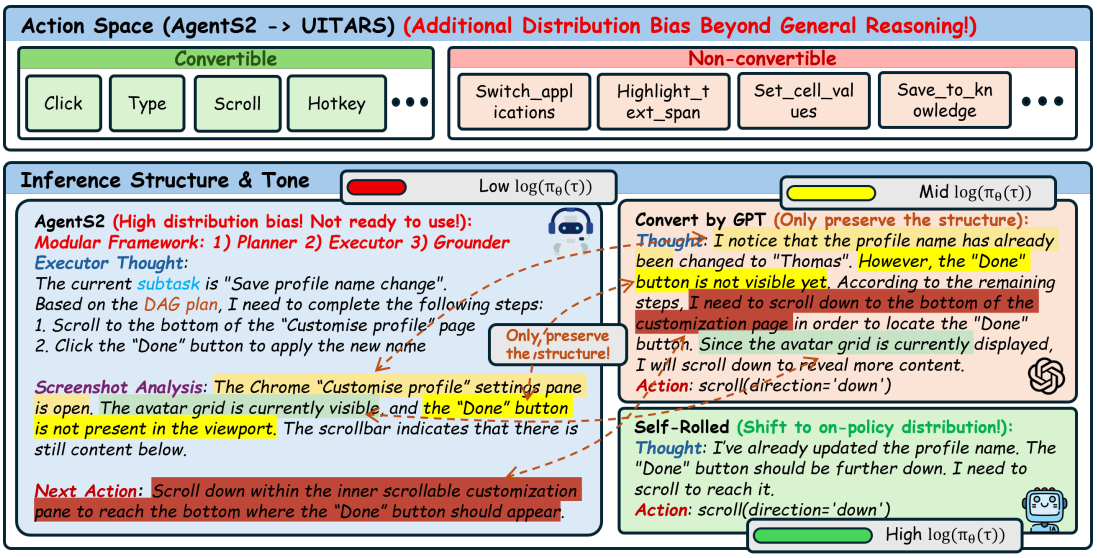


Figure 2: **Distribution bias from framework experts to end-to-end GUI policies. Top: action-space mismatch** (Agent S2 includes non-convertible framework actions beyond UI-TARS primitives). **Bottom: inference structure mismatch.** Agent S2 traces (left) interleave multi-role artifacts (planner/executor/grounder), yielding low likelihood under the base policy (low $\log \pi_\theta(\tau)$). Naive conversion (middle) maps format but largely preserves the framework structure, remaining off-manifold. Self-rolled execution (right) re-generates actions under the base policy (plan-conditioned), producing more policy-compatible, higher-likelihood trajectories.

$$\hat{\mathcal{T}}_x = \{\tau_x^{\text{off}}\} \cup \{\tau_{x,2}, \ldots, \tau_{x,N}\}. \tag{3}$$

This injection guarantees that even in failed batches, the optimizer receives at least one positive signal. Given the finalized group $\hat{\mathcal{T}}_x$, we compute advantages using group-wide normalization. For $\tau_\ell \in \hat{\mathcal{T}}_x$ with reward set $\hat{G} = \{R(\tau) : \tau \in \hat{\mathcal{T}}_x\}$:

$$\hat{A}_\ell = \frac{R(\tau_\ell) - \text{mean}(\hat{G})}{\text{std}(\hat{G}) + \epsilon}. \tag{4}$$

The BEPA objective extends GRPO to accommodate the mixed composition of $\hat{\mathcal{T}}_x$:

$$\mathcal{J}_{\text{BEPA}}(\theta) = \frac{1}{Z} \sum_{\tau \in \hat{\mathcal{T}}_x} \sum_{t=1}^{|\tau|} L^{\text{CLIP}}(r_t, \hat{A}_\tau), \tag{5}$$

where $Z = \sum_\tau |\tau|$, $L^{\text{CLIP}}(r, A) = \min[rA,\ \text{clip}(r; 1 \pm \epsilon)A]$, and $r_t = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ is the importance ratio. The clipping empirically keeps π_θ within a trust region of $\pi_{\theta_{\text{old}}}$ (Schulman et al., 2015). The convergence analysis of the resulting mixed-policy GRPO objective is provided in Appendix B.

5 Experiments

5.1 Experiment Setup

Dataset & Benchmarks. We conduct RLVR training on the OSWorld-Verified (Xie et al., 2024) and assess cross-domain generalization on MMBench-GUI (Wang et al., 2025c) and Online-Mind2Web (Deng et al., 2023a; Xue et al., 2025). **OSWorld-Verified** comprises 369 real-world tasks requiring file I/O and multi-app workflows. Training RL agents on such complex benchmarks is often impeded by extremely sparse reward signals, as many tasks remain effectively unsolvable by current models, yielding negligible feedback for optimization. To mitigate this and ensure **informative reward signals** during exploration, we curate a subset of "high-value" tasks that offer verifiable success trajectories. Specifically, we identify tasks solvable by *either* the external expert (Agent S2) *or* the base policy (UITARS1.5-7B Pass@5), resulting in a pool of 150 tasks where valid supervision is guaranteed. From this pool, we randomly sample 80% (120 tasks) and 8 other tasks to form the **Training Set** $\mathcal{D}_{\text{train}}$ (128 tasks), ensuring the model focuses on learnable behaviors rather than stalling on intractable scenarios. The remaining 20% of the solvable pool, combined with other tasks, constitutes the **Held-out Test Set** $\mathcal{D}_{\text{held\_out}}$ (241 tasks), which is used to strictly evaluate the agent's generalization capability to unseen scenarios. Detailed benchmark information is provided in the Appendix C.

Baselines. We evaluate our approach against a comprehensive suite of competitive baselines span-

ning multiple categories: 1) **closed-source LLMs** with strong reasoning capabilities, 2) **open-source GUI agents** specialized for visual interaction, and 3) **modular agent frameworks** that employ compositional strategies for computer use tasks. 4) **training methodologies**, we benchmark against GRPO (Shao et al., 2024) as the pure on-policy baseline, along with several **expert integration methods** that incorporate converted expert traces $\mathcal{D}_{\text{conv}}$ (the conversion prompt is in Appendix F.3): SFT, RL w/ SFT Loss (incorporating SFT loss during RL training), and SFT+RL (a two-stage process continuing RL after SFT). We also compare with LUFFY (Yan et al., 2025), which augments RLVR with *raw off-policy reasoning traces* via Mixed-Policy GRPO and policy shaping, and Trace Replacement, which differs from LUFFY by replacing trajectories with off-policy traces only upon group failure and calculating the importance ratio using the old policy likelihood rather than a constant (BEPA is based on the trace replacement). See more details in Appendix D.

Distribution Analysis Protocol. To measure internalization beyond the base policy's existing competence, we construct an *expert-only* task set $\mathcal{D}_{\text{expert\_only}}$ by removing all tasks that the untrained base policy can solve with Pass@5 from the self-rolled success pool $\mathcal{D}_{\text{self}}$, i.e., $\mathcal{D}_{\text{expert\_only}} = \{(x,\tau) \in \mathcal{D}_{\text{self}} \mid \text{Pass@5}_{\text{base}}(x) = 0\}$, yielding $|\mathcal{D}_{\text{expert\_only}}| = 54$. We then form an update-triggered subset $\mathcal{D}_{\text{upd}} \subset \mathcal{D}_{\text{expert\_only}}$ of 19 tasks on which BEPA triggers cache updates during training. On $\mathcal{D}_{\text{upd}}$, we score token probabilities under the base policy, aggregate histograms over $[0,1]$, and in Figure 3 report two descriptive statistics: the tail mass $\Pr(p{<}0.2)$ (showing how much probability remains in LUFFY's shaping band after conversion) and the Jensen–Shannon divergence to the on-policy reference (Menéndez et al., 1997) (a coarse measure of shape similarity to on-policy rollouts rather than a formal on-/off-policy test).

Implementation Details. We use UITARS1.5-7B as the base end-to-end agent for all experiments. The plan extractor ϕ we use is GPT-4o. As the expert source, we collect 115 successful OSWorld trajectories generated by Agent S2 and use them for conversion and self-rolling (success rate is 76%, resulting in $|\mathcal{D}_{\text{self}}| = 88$). For reinforcement learning, we adopt GRPO with a rollout group size of $N = 8$ and a maximum episode length of 15 steps, following the OSWorld's 15-step evaluation setting.

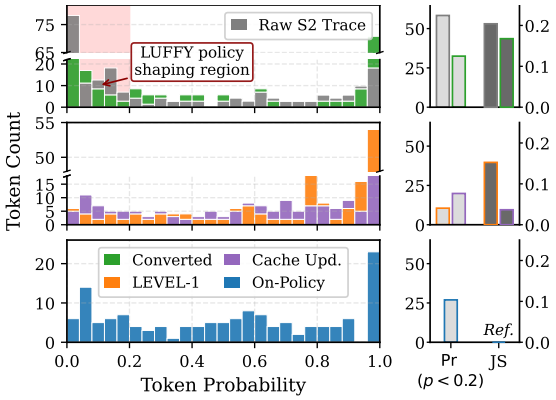


Figure 3: **Token-probability distributions and simple diagnostics on $\mathcal{D}_{\text{upd}}$. Top**: Raw Agent S2 traces (gray) versus converted traces (green). Conversion substantially reduces the extreme low-probability tail ($p < 0.2$) but still leaves a large mass in regions atypical for the base policy. **Middle**: LEVEL-1 self-rolled guidance (orange) and LEVEL-2 cache-updated traces (purple). LEVEL-1 makes guidance highly reachable (peaked near $p \approx 1$), while LEVEL-2 reshapes the cache toward the on-policy reference, yielding a much smaller JS divergence. **Bottom**: On-policy baseline. **Right**: Tail mass $\Pr(p < 0.2)$ and JS w.r.t. the on-policy reference.

See more details in the Appendix E.

5.2 Main Results

Overall Performance. As shown in Table 1, BEPA achieves 32.13% success on OSWorld-Verified, improving over UITARS1.5-7B (22.87%) by +9.26 points (+40.5% relative) and over GRPO (23.60%) by +8.53 points (+36.1% relative). BEPA also improves all three splits: on $\mathcal{D}_{\text{train}}$ it reaches 73.23% (vs. 55.12% for UITARS1.5-7B and 58.02% for GRPO), and on the strictly held-out set $\mathcal{D}_{\text{held\_out}}$ it improves to 10.30% (vs. 5.74% for UITARS1.5-7B and 5.32% for GRPO), indicating better generalization beyond solvable training tasks. Detailed results on OSWorld-Verified, MMBench-GUI and Online-Mind2Web are provided in Appendix G.

Baseline Analysis. Naïve expert integration strategies consistently underperform. SFT+RL drops to 14.74% overall and 39.37% on $\mathcal{D}_{\text{train}}$, indicating severe forgetting and weakened exploration. RL+SFT reaches 20.88% overall and does not improve held-out generalization (3.53% on $\mathcal{D}_{\text{held\_out}}$). Trace Replacement (23.91%) and LUFFY (24.11%) remain close to GRPO overall, while showing a large generalization gap (Replacement: 66.50% $\rightarrow$ 1.29%; LUFFY: 65.44% $\rightarrow$ 2.16% from $\mathcal{D}_{\text{train}}$ to $\mathcal{D}_{\text{held\_out}}$), suggesting that *static* off-policy injection does not robustly trans-

Method	$\mathcal{D}_{\text{expert\_only}}$	$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{held\_out}}$	Overall (%)
Agent S2	-	-	-	33.00
Jedi-7B w/ o3	-	-	-	**42.40**
OpenAI o3	-	-	-	9.10
Doubao-1.5-Think	-	-	-	31.90
Claude-4-Sonnet	-	-	-	31.20
UITARS-72B-DPO	-	-	-	24.00
GUI-Owl-7B	-	-	-	**32.10**
OpenCUA-7B	-	-	-	$24.30_{\pm 1.40}$
ARPO	-	-	-	$23.86_{\pm 0.72}$
UITARS1.5-7B	18.52	55.12	5.74	$22.87_{\pm 0.97}$
SFT	5.56	47.77	1.65	$17.65_{\pm 0.55}$
GRPO	11.11	58.02	5.32	$23.60_{\pm 1.15}$
RL+SFT	14.81	53.55	3.53	$20.88_{\pm 1.55}$
SFT+RL	9.26	39.37	1.66	$14.74_{\pm 0.60}$
Trace Replacement	18.52	66.50	1.29	$23.91_{\pm 2.35}$
LUFFY	19.01	65.44	2.16	$24.11_{\pm 2.10}$
LEVEL-1	25.93	69.20	5.05	$27.30_{\pm 1.45}$
LEVEL-2	29.18	71.65	7.48	$29.74_{\pm 0.90}$
BEPA (ours)	35.19	73.23	10.30	$\mathbf{32.13}_{\pm 0.25}$

Table 1: **Performance on OSWorld-Verified.** We reported the average success rate (%) on $\mathcal{D}_{\text{expert\_only}}$, $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{held\_out}}$. For the performance on the whole OSWorld-Verified dataset (Overall), the average success rate (%) with standard deviation across 3 runs is reported.

fer to unseen tasks. Moreover, LUFFY performs similarly to Replacement in our setting because its policy shaping mainly reweights gradients toward low-probability tokens in *raw* off-policy traces; on $\mathcal{D}_{\text{upd}}$, raw Agent S2 traces place 58.33% of tokens in the $p < 0.2$ band with JS≈ 0.2040 to the on-policy reference, whereas after conversion the tail mass and JS drop to 32.48% and 0.1676 (Figure 3), leaving a much smaller shaping band for LUFFY to act on.

5.3 Why Static Expert Integration Fails

SFT vs. GRPO in Dynamic GUIs. OSWorld-style tasks are highly dynamic: the initial state and valid actions can change across episodes due to human-verification popups, time-sensitive banners, and other transient UI elements (Figure 4). SFT minimizes token-level cross-entropy on a small converted expert dataset (125 successful tasks, 1070 step-level pairs), encouraging the policy to *replay* expert action sequences under an idealized history rather than re-reading the current screen and adapting to these variations. By contrast, GRPO-style mixed updates use group-normalized advantages (Eqs. (4) and (5)) and PPO clipping to simultaneously push up successful trajectories, push down failures, and keep updates within a trust region of $\pi_{\theta_{\text{old}}}$, which stabilizes learning on mixed on-/off-policy batches. Empirically, SFT does increase the log-probability of converted traces (Figure 5b), but higher off-policy likelihood alone does not translate into robust on-policy success under distribution shift: SFT+RL collapses entropy during the SFT stage and struggles to recover exploration (Figure 5a), and RL+SFT yields only modest overall gains and weak held-out performance (Table 1).

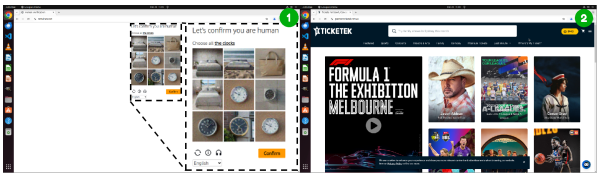


Figure 4: **Dynamic environments in OSWorld.** (1) **Pop-ups**: human-verification challenges and pop-up dialogs appear stochastically with the random content. (2) **Web Content**: Real time content and random ads.

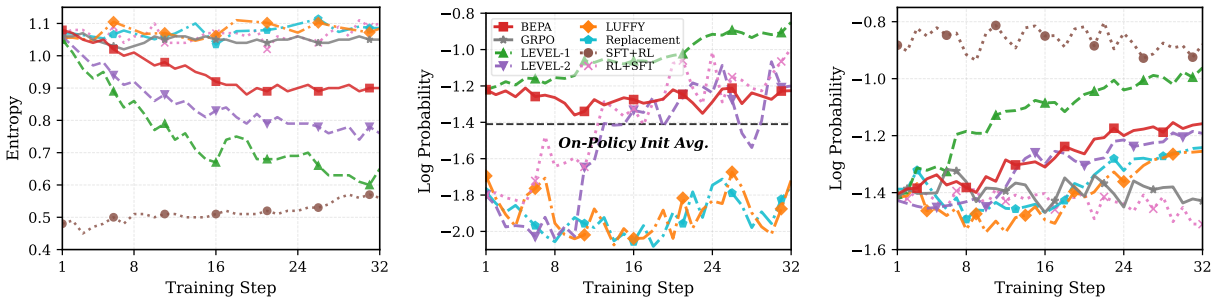
Static off-policy mixing. For LUFFY and naive Replacement, the main issue is not exploration collapse—both maintain entropy close to the on-policy baseline (Figure 5a)—but *non-internalization* under distribution shift. They improve off-policy successes only marginally (Figure 5b) and can initially hinder on-policy learning, where success log-probability dips before partially recovering to a low plateau (Figure 5c), and held-out performance remains low (Table 1). This indicates that static guidance is not kept aligned with the policy's evolving manifold.

5.4 Mechanism Analysis

We analyze why BEPA works beyond final success rates through two complementary mechanisms: (i) **native, learnable guidance** from LEVEL-1; and (ii) **dynamic alignment** from LEVEL-2.

5.4.1 LEVEL-1: Native Guidance

LEVEL-1 makes expert solutions *reachable* by regenerating trajectories under the base policy with plan conditioning, avoiding the direct mismatch of compositional expert traces (Figure 2). On $\mathcal{D}_{\text{upd}}$, raw Agent S2 traces exhibit a very heavy low-probability tail and noticeable divergence to the on-policy reference (tail mass 58.33%, JS 0.2040), and conversion alone only partially mitigates this (converted: 32.48%, JS 0.1676; Figure 3). LEVEL-1 self-rolled guidance further increases the base policy's confidence on guided successes (tail mass 10.53%, JS 0.1525) and achieves higher average log-probability than both converted traces and initial on-policy rollouts (Figure 5b), making guidance much more learnable. Empirically, this yields a strong boost on expert-covered tasks (25.93%



(a) Entropy (b) Off-policy Avg. Log-prob. (c) On-policy Avg. Log-prob.

Figure 5: **Training dynamics.** (a) **Policy entropy**: SFT+RL collapses during the SFT stage, and LEVEL-1 shows a pronounced entropy drop due to high-confidence plan-conditioned self-rolled guidance; in contrast, LUFFY/Replacement maintain entropy close to the on-policy baseline, while BEPA sustains a moderate entropy profile alongside steadily improving expert-only success. (b–c) **Avg. log-probability on successful trajectories**: BEPA improves off-policy and on-policy successes in tandem, indicating gradual assimilation without degrading on-policy learning; LUFFY/Replacement exhibit an early dip and only partial recovery on on-policy log-probability, converging to a low plateau, while SFT+RL shows overfitting that harms on-policy improvement.

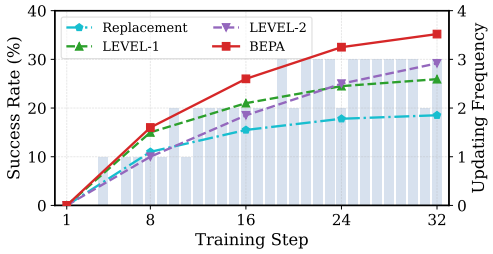


Figure 6: **Expert-to-Policy Assimilation.** The evolution of expert-only success ($\mathcal{D}_{\text{expert\_only}}$, left axis) versus cache updating frequency (right axis). The updating frequency of BEPA rises in tandem with performance, demonstrating that rising competence drives active cache updates, thereby sustaining the effective **assimilation** of expert guidance into the policy.

on $\mathcal{D}_{\text{expert}}$), but limited transfer to held-out tasks (5.05% on $\mathcal{D}_{\text{held\_out}}$) and a pronounced entropy drop (Figure 5a and Table 1), indicating that overly confident plan-conditioned guidance can narrow exploration if left static.

5.4.2 LEVEL-2: Dynamic Alignment

LEVEL-2 keeps guidance synchronized with the evolving policy by refreshing the cache with the agent's own successful executions (Sec. 4.2), instead of repeatedly injecting a fixed converted distribution as in static Replacement/LUFFY. On $\mathcal{D}_{\text{upd}}$, this yields guidance whose token-probability histogram is much closer to the on-policy reference: the JS divergence is reduced to 0.0366, and the tail mass $\Pr(p < 0.2)$ moves from 10.53% (LEVEL-1) toward the on-policy profile (19.88% vs. 26.95%; Figure 3). LEVEL-2 also improves held-out generalization to 7.48%, and BEPA further raises it to 10.30% on $\mathcal{D}_{\text{held\_out}}$ (Table 1), while guided and on-policy success likelihoods improve in tandem (Figures 5b and 5c), indicating that guidance is being gradually internalized rather than remaining a static off-policy signal.

5.5 Ablation

The ablations in Table 1 and the learning curves in Figure 6 match this division of labor. Replacement saturates early on expert-covered tasks (18.52% on $\mathcal{D}_{\text{expert}}$) and generalizes poorly (1.29% on $\mathcal{D}_{\text{held\_out}}$). LEVEL-1 improves reachability and rises faster on $\mathcal{D}_{\text{expert}}$ (25.93%) but plateaus with limited transfer. LEVEL-2 delivers sustained improvements (29.18% on $\mathcal{D}_{\text{expert}}$; 7.48% held-out). BEPA combines both, achieving the best expert-covered success (35.19%) and held-out generalization (10.30%); moreover, Figure 6 shows its cache update frequency increases in tandem with expert-only success, directly evidencing self-aligned assimilation.

6 Conclusion

We identify expert–policy mismatch as a central obstacle to turning high-quality framework trajectories into reliable gains for end-to-end GUI policies under RLVR. BEPA addresses this with a bi-level assimilation scheme: LEVEL-1 converts alien expert traces into high-confidence, policy-compatible self-rolled trajectories, while LEVEL-2 keeps guidance aligned with the evolving policy via a self-updating cache that is injected only when on-policy exploration fails. Overall, our results suggest that heterogeneous expert data is most effective when progressively assimilated into the learner's own distribution, rather than statically mixed at the loss level.

7 Limitations

While BEPA delivers consistent gains under the evaluated setups, our study instantiates BEPA only on GUI-based computer-use benchmarks, using a specific backbone together with Agent S2 and GUI-Owl as expert sources and a particular plan-extraction pipeline. Extending BEPA to other domains and interaction modalities (e.g., mobile platforms, productivity suites, or non-GUI environments), as well as to more diverse and automatically mined expert pools, is a natural direction for future work. Our experiments further adopt an RLVR regime with sparse, verifiable rewards; adapting cache refresh and conditional guidance injection to settings with noisier or preference-based feedback, and combining BEPA with alternative reward modeling and credit-assignment schemes, are promising next steps. Finally, BEPA introduces additional machinery in the form of self-rolled trajectories and per-task caches; exploring lighter-weight variants, tighter integration with existing agent frameworks, and broader evaluations across backbones and longer-horizon tasks will help further characterize the scalability and generality of bi-level expert-to-policy assimilation.

References

Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. 2024. Agent-e: From autonomous web navigation to foundational design principles in agentic systems. *arXiv preprint arXiv:2407.13032*.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations (ICLR)*.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Adept AI. 2022. Act-1: Transformer for actions. https://www.adept.ai/blog/act-1. Blog post.

LaVague AI. 2025. Lavague: Large action model framework. https://github.com/lavague-ai/LaVague. GitHub repository.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. 2025. Reinforcement learning for long-horizon interactive llm agents. *Preprint*, arXiv:2502.01600.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, R. Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 179 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023a. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023b. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, and 1 others. 2024. AssistGUI: Task-oriented pc graphical user interface automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *International Conference on Learning Representations (ICLR)*.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, Jingji Chen, Jingjia Huang, Kang Lei, Liping Yuan, Lishu Luo, Pengfei Liu, Qinghao Ye, Rui Qian, Shen Yan, and 178 others. 2025. Seed1.5-vl technical report. *arXiv preprint arXiv:2505.07062*.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, X. Zhang, and H. Shum. 2025. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. 2025. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*. Under review at ICLR 2026.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8925–8948. Association for Computational Linguistics.

Rong Jin, Xiangheng Li, and Tong Zhang. 2024. On stationary point convergence of ppo-clip. In *Proceedings of the International Conference on Learning Representations*, ICLR.

Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations (ICLR)*.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. *Preprint*, arXiv:2503.20783.

I. Loshchilov and F. Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. 2025. Arpo:end-to-end policy optimization for gui agents with experience replay. *Preprint*, arXiv:2505.16282.

Jiantao Mei, Yu Bai, Chi Jin, and Jason Lee. 2020. On the global convergence rates of softmax policy gradient methods. In *Proceedings of the 37th International Conference on Machine Learning*, ICML.

M. Menéndez, J. A. Pardo, L. Pardo, and M. Pardo. 1997. The jensen-shannon divergence. *Journal of The Franklin Institute-engineering and Applied Mathematics*, 334:307–318.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, and 1 others. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Magnus Müller and Gregor Žunič. 2024. Browser use: Enable ai to control your browser. `https://github.com/browser-use/browser-use`. GitHub repository.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Md. Mehrab Tanjim, and 10 others. 2024. Gui agents: A survey. In *Findings of the Association for Computational Linguistics (ACL)*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, P. Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Rui Qian, Xin Yin, Chuanhang Deng, Zhiyuan Peng, Jian Xiong, Wei Zhai, and Dejing Dou. 2025. Uground: Towards unified visual grounding with unrolled transformers. *arXiv preprint arXiv:2510.03853*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Rafael Rafailov, Archit Sharma, E. Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Sashank J. Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczos, and Alexander Smola. 2016. Stochastic variance reduction for nonconvex optimization. In *Proceedings of the 33rd International Conference on Machine Learning*, ICML, pages 314–323. PMLR.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. In *4th International Conference on Learning Representations*.

John Schulman, S. Levine, P. Abbeel, Michael I. Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, R. Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, Rotterdam, The Netherlands. ACM.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan

Chen, Taiwei Shi, Ran Xu, and Caiming Xiong. 2025. Coact-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Bo Li, Chen Dun, Chong Liu, and 87 others. 2025a. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Weinan Gan, Xingshan Zeng, Shuai Yu, Xinlong Hao, Kun Shao, Yasheng Wang, and Ruiming Tang. 2024. Gui agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Bo Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, and 23 others. 2025b. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*.

Xuehui Wang, Zhenyu Wu, Jingjing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, Weiyun Wang, Xiangyu Zhao, Jixuan Chen, Haodong Duan, Tianbao Xie, Chenyu Yang, Shiqian Su, Yue Yu, Yuan Huang, and 9 others. 2025c. Mmbench-gui: Hierarchical multi-platform evaluation framework for gui agents. *arXiv preprint arXiv:2507.19478*.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. Scaling computer-use grounding via user interface decomposition and synthesis. *Preprint*, arXiv:2505.13227.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, T. Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Wei Xiong, Hanze Dong, Chen Ye, Han Zhong, Nan Jiang, and Tong Zhang. 2024. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. In *International Conference on Machine Learning (ICML)*.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. Aguvis: Unified pure vision agents for autonomous gui interaction. In *International Conference on Machine Learning (ICML)*.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. 2025. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Ming-Jie Ma, Qingwei Lin, S. Rajmohan, Dongmei Zhang, and Qi Zhang. 2024. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. 2025. Bread: Branched rollouts from expert anchors bridge sft & rl for reasoning. *arXiv preprint arXiv:2506.17211*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*.

Daniel M. Ziegler, Nisan Stiennon, Jeff Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and G. Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.

Appendix Contents

A BEPA Algorithm Details

A.1 Pseudo-code

Algorithm 1 LEVEL-1: Self-Rolled Execution (Seed Cache Initialization)

Require: Expert successful traces $\mathcal{D}_E = \{(x, \tau_x^E)\}$; plan extractor $\phi(\cdot)$; base policy π_θ; verifier $R(\cdot)$
Ensure: Self-rolled seed set $\mathcal{D}_{\text{self}}$; initialized cache $\mathcal{E}$

$\mathcal{D}_{\text{self}} \leftarrow \emptyset$; $\mathcal{E} \leftarrow \emptyset$
for all $(x, \tau_x^E) \in \mathcal{D}_E$ **do**
  $p_x \leftarrow \phi(\tau_x^E)$
  Roll out $\pi_\theta(\cdot \mid s_t, x, p_x)$ in the environment to obtain τ_x^{self}
  if $R(\tau_x^{\text{self}}) = 1$ **then**
    $\mathcal{D}_{\text{self}} \leftarrow \mathcal{D}_{\text{self}} \cup \{(x, \tau_x^{\text{self}})\}$
    $\mathcal{E}_x \leftarrow \tau_x^{\text{self}}$ ▷ seed cache with policy-compatible successes
  end if
end for
return $\mathcal{D}_{\text{self}}, \mathcal{E}$

Algorithm 2 LEVEL-2: Self-Aligned Off-Policy Assimilation with Conditional Trace Replacement

Require: Task set $\mathcal{X}$; policy parameters θ; cache $\mathcal{E}$ (seeded by $\mathcal{D}_{\text{self}}$); verifier $R(\cdot)$
Require: Group size N; cache update rule $\mathcal{U}(\cdot)$
Ensure: Updated policy parameters θ

for iteration $k = 1, 2, \ldots$ **do**
  Set behavior policy $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$ ▷ standard on-policy collection in GRPO
  for all sampled tasks $x \sim \mathcal{X}$ **do**
    Collect a rollout group $\mathcal{T}_x = \{\tau_{x,i}\}_{i=1}^N$ by sampling $\tau_{x,i} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$
    $\mathcal{T}_x^{\text{succ}} \leftarrow \{\tau \in \mathcal{T}_x \mid R(\tau) = 1\}$
    if $\mathcal{T}_x^{\text{succ}} \neq \varnothing$ **then**
      $\tau_x^{\text{new}} \leftarrow \mathcal{U}(\mathcal{T}_x^{\text{succ}})$ ▷ e.g., random sampling
      $\mathcal{E}_x \leftarrow \tau_x^{\text{new}}$ ▷ dynamic cache refresh (Sec. 4.2)
      $\hat{\mathcal{T}}_x \leftarrow \mathcal{T}_x$
    else
      if $\mathcal{E}_x \neq \varnothing$ **then**
        $\tau_x^{\text{off}} \leftarrow \mathcal{E}_x$
        $\hat{\mathcal{T}}_x \leftarrow \{\tau_x^{\text{off}}\} \cup \{\tau_{x,2}, \ldots, \tau_{x,N}\}$ ▷ Eq. (3)
      else
        $\hat{\mathcal{T}}_x \leftarrow \mathcal{T}_x$
      end if
    end if
    Compute $\hat{A}$ by Eq. (4)
  end for
  Update θ by maximizing $\mathcal{J}_{\text{BEPA}}(\theta)$ in Eq. (5)
end for

A.2 Plan Concatenation Example

Plan-Conditioned Prompt Example

```
Can you enable the 'Do Not Track' feature
    in Chrome to enhance my online privacy?

1. Click on the three-dot menu icon in the
    upper-right corner of the Chrome
    window.
2. Select "Settings" from the dropdown
    menu.
3. Click on "Privacy and security".
4. Scroll down to find the "Send a 'Do Not
    Track' request..." option.
5. Enable the toggle to send a 'Do Not
    Track' request.
6. Click "Confirm" to apply the changes.
```

Figure 7: An example of plan conditioning for LEVEL-1: we append an extracted plan to the original instruction.

B Convergence Analysis

We analyze BEPA as a clipped mixed-policy policy-gradient method based on Eqs. (4) and (5), and show that it enjoys the standard $O(1/\sqrt{K})$ non-convex convergence rate for the GRPO/PPO surrogate.

B.1 Setup and Assumptions

Let $r_t(\theta)$ and $w_t(\theta)$ be the usual ratio and clipped ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t, x)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t, x)}, \qquad w_t(\theta) = \text{clip}\big(r_t(\theta);\, 1-\epsilon,\, 1+\epsilon\big). \tag{6}$$

At iteration k, BEPA optimizes the clipped GRPO surrogate

$$\mathcal{J}_{\text{BEPA}}(\theta) = \mathbb{E}_{\tau\sim\mu_k}\left[\frac{1}{Z}\sum_{t=1}^{|\tau|} L^{\text{CLIP}}\big(r_t(\theta), \hat{A}_\tau\big)\right], \tag{7}$$

where $Z = \sum_\tau |\tau|$ and $L^{\text{CLIP}}(r, A) = \min[rA, \text{clip}(r; 1\pm\epsilon)A]$. The expectation is taken with respect to the behavior distribution μ_k induced by BEPA at iteration k (a mixture of on-policy rollouts and cached trajectories; cf. Sec. 4.3). For a trajectory τ and parameters θ, define the per-trajectory gradient contribution

$$H_k(\tau;\theta) := \frac{1}{Z}\sum_t w_t(\theta)\,\hat{A}_\tau\,\nabla_\theta \log \pi_\theta(a_t \mid s_t, x), \tag{8}$$

so the stochastic gradient used by BEPA is $\hat{g}(\theta_k) = H_k(\tau;\theta_k)$ with $\tau \sim \mu_k$.
We adopt standard assumptions used in non-convex policy-gradient and PPO analyses (Reddi et al., 2016; Mei et al., 2020; Jin et al., 2024):

(i) *L-smoothness.* $\mathcal{J}_{\text{BEPA}}(\theta)$ has L-Lipschitz gradients: $\|\nabla\mathcal{J}_{\text{BEPA}}(\theta) - \nabla\mathcal{J}_{\text{BEPA}}(\theta')\|_2 \le L\|\theta-\theta'\|_2$.

(ii) *Bounded score and advantage.* There exist constants $G, A_{\max} > 0$ such that for all (s_t, x, a_t) and all τ, $\|\nabla_\theta \log \pi_\theta(a_t \mid s_t, x)\|_2 \le G$ and $|\hat{A}_\tau| \le A_{\max}$. In our setting, rewards are $0/1$ and group-normalized advantages are computed by Eq. (4) over a rollout group of size N. When trace replacement triggers (all on-policy rollouts fail and the task has a cached success), the reward multiset is $\hat{G} = \{1, 0, \ldots, 0\}$, which yields $\hat{A}_{\text{succ}} = \sqrt{N-1}$ and $\hat{A}_{\text{fail}} = -1/\sqrt{N-1}$, so we may take $A_{\max} = \sqrt{N-1}$ (e.g., $\sqrt{7}$ for N=8). In all other cases, rewards lie in $\{0, 1\}$ and Eq. (4) produces advantages with smaller magnitude.

(iii) *Clipped importance weights.* By construction, $|w_t(\theta)| \le 1+\epsilon$ for all t, θ.

Unbiasedness for the surrogate. We emphasize that we analyze the clipped surrogate $\mathcal{J}_{\text{BEPA}}$ in Eq. (7), not the unclipped environment return. At iteration k, μ_k is fixed by the data-collection procedure (using $\pi_{\theta_{\text{old}}}$ and the current cache), so

$$\mathcal{J}_{\text{BEPA}}(\theta) = \mathbb{E}_{\tau\sim\mu_k}\big[F(\theta,\tau)\big], \qquad F(\theta,\tau) := \frac{1}{Z}\sum_t L^{\text{CLIP}}\big(r_t(\theta), \hat{A}_\tau\big). \tag{9}$$

Differentiating under the expectation gives

$$\nabla\mathcal{J}_{\text{BEPA}}(\theta) = \mathbb{E}_{\tau\sim\mu_k}\big[\nabla_\theta F(\theta,\tau)\big] = \mathbb{E}_{\tau\sim\mu_k}\big[H_k(\tau;\theta)\big], \tag{10}$$

so $\hat{g}(\theta_k) = H_k(\tau;\theta_k)$ is an unbiased estimator of $\nabla\mathcal{J}_{\text{BEPA}}(\theta_k)$ *for any* behavior distribution μ_k, including mixtures with injected cached trajectories. This is exactly the same notion of unbiasedness used in standard policy-gradient theory, but applied to the clipped surrogate objective: we do not claim monotonic improvement guarantees for the true environment return.

B.2 Variance Bound

Using (ii)–(iii) and Cauchy–Schwarz, for any trajectory τ and iteration k we have

$$\begin{aligned}\|H_k(\tau;\theta)\|_2^2 &= \left\| \frac{1}{Z} \sum_t w_t(\theta)\, \hat{A}_\tau\, \nabla_\theta \log \pi_\theta(a_t \mid s_t, x) \right\|_2^2 \\ &\le \frac{1}{Z^2} \sum_t |w_t(\theta)|^2\, |\hat{A}_\tau|^2\, \big\|\nabla_\theta \log \pi_\theta(a_t \mid s_t, x)\big\|_2^2 \\ &\le \frac{1}{Z^2} \sum_t (1+\epsilon)^2 A_{\max}^2 G^2 \;\le\; \sigma^2, \end{aligned} \tag{11}$$

for some $\sigma^2 = O((1+\epsilon)^2 G^2 A_{\max}^2)$ that does not depend on the behavior distribution μ_k. Taking expectation over $\tau \sim \mu_k$ gives a uniform second-moment bound

$$\mathbb{E}_{\tau\sim\mu_k}\big[\|\hat{g}(\theta_k)\|_2^2\big] = \mathbb{E}_{\tau\sim\mu_k}\big[\|H_k(\tau;\theta_k)\|_2^2\big] \le \sigma^2, \tag{12}$$

for all iterations k, regardless of how μ_k mixes on-policy and cached trajectories.

B.3 Convergence of BEPA

Theorem 1 (Convergence of BEPA). *Let $\{\theta_k\}_{k=0}^{K-1}$ be the parameter sequence generated by BEPA with learning rate $\alpha_k = c/\sqrt{K}$. Under assumptions* (i)–(iii)*, there exist constants $C_1, C_2 > 0$ such that*

$$\begin{aligned}\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}\big[\|\nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)\|_2^2\big] \;\le\; & \frac{C_1}{\sqrt{K}}\big(\mathcal{J}_{\mathrm{BEPA}}(\theta^*) - \mathcal{J}_{\mathrm{BEPA}}(\theta_0)\big) \\ & + \frac{C_2}{\sqrt{K}}\,\sigma^2, \end{aligned} \tag{13}$$

where θ^* is an optimal solution of the surrogate objective. Thus BEPA converges to a first-order stationary point of $\mathcal{J}_{\mathrm{BEPA}}(\theta)$ at rate $O(1/\sqrt{K})$.

Proof sketch. By L-smoothness of $\mathcal{J}_{\mathrm{BEPA}}(\theta)$ and the update $\theta_{k+1} = \theta_k - \alpha_k \hat{g}(\theta_k)$, we have

$$\mathcal{J}_{\mathrm{BEPA}}(\theta_{k+1}) \le \mathcal{J}_{\mathrm{BEPA}}(\theta_k) - \alpha_k \big\langle \nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k), \hat{g}(\theta_k)\big\rangle + \frac{L\alpha_k^2}{2}\,\|\hat{g}(\theta_k)\|_2^2. \tag{14}$$

Taking expectation and using $\mathbb{E}[\hat{g}(\theta_k)] = \nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)$ gives

$$\begin{aligned}\mathbb{E}[\mathcal{J}_{\mathrm{BEPA}}(\theta_{k+1})] \le \mathbb{E}[\mathcal{J}_{\mathrm{BEPA}}(\theta_k)] - \alpha_k\, \mathbb{E}\big[\|\nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)\|_2^2\big] \\ + \frac{L\alpha_k^2}{2}\,\mathbb{E}\big[\|\hat{g}(\theta_k)\|_2^2\big]. \end{aligned} \tag{15}$$

Applying the variance bound Eq. (12) and rearranging yields

$$\alpha_k\, \mathbb{E}\big[\|\nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)\|_2^2\big] \;\le\; \mathbb{E}[\mathcal{J}_{\mathrm{BEPA}}(\theta_k)] - \mathbb{E}[\mathcal{J}_{\mathrm{BEPA}}(\theta_{k+1})] + C\,\alpha_k^2 \sigma^2, \tag{16}$$

for some constant $C > 0$ (e.g., $C = L/2$). Summing Eq. (16) over $k = 0, \ldots, K-1$ gives

$$\begin{aligned}\sum_{k=0}^{K-1} \alpha_k\, \mathbb{E}\big[\|\nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)\|_2^2\big] &\le \mathbb{E}[\mathcal{J}_{\mathrm{BEPA}}(\theta_0)] - \mathbb{E}[\mathcal{J}_{\mathrm{BEPA}}(\theta_K)] + C\,\sigma^2 \sum_{k=0}^{K-1} \alpha_k^2 \\ &\le \mathcal{J}_{\mathrm{BEPA}}(\theta_0) - \mathcal{J}_{\mathrm{BEPA}}(\theta^*) + C\,\sigma^2 \sum_{k=0}^{K-1} \alpha_k^2, \end{aligned} \tag{17}$$

since $\mathcal{J}_{\mathrm{BEPA}}(\theta_K) \geq \mathcal{J}_{\mathrm{BEPA}}(\theta^*)$. With constant step size $\alpha_k = \alpha = c/\sqrt{K}$ we have

$$\sum_{k=0}^{K-1} \alpha_k = K\alpha = c\sqrt{K}, \qquad \sum_{k=0}^{K-1} \alpha_k^2 = K\alpha^2 = c^2. \tag{18}$$

Dividing Eq. (17) by $\sum_{k=0}^{K-1} \alpha_k$ and using Eq. (18) yields

$$\begin{aligned} \frac{\sum_{k=0}^{K-1} \alpha_k \mathbb{E}\big[\|\nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)\|_2^2\big]}{\sum_{k=0}^{K-1} \alpha_k} &\leq \frac{\mathcal{J}_{\mathrm{BEPA}}(\theta_0) - \mathcal{J}_{\mathrm{BEPA}}(\theta^*)}{c\sqrt{K}} + \frac{C\,\sigma^2 c^2}{c\sqrt{K}} \\ &= \frac{C_1}{\sqrt{K}}\big(\mathcal{J}_{\mathrm{BEPA}}(\theta_0) - \mathcal{J}_{\mathrm{BEPA}}(\theta^*)\big) + \frac{C_2}{\sqrt{K}}\,\sigma^2, \end{aligned} \tag{19}$$

for suitable constants $C_1, C_2 > 0$. Since α_k is constant, the left-hand side is simply the average gradient norm:

$$\frac{\sum_{k=0}^{K-1} \alpha_k \mathbb{E}\big[\|\nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)\|_2^2\big]}{\sum_{k=0}^{K-1} \alpha_k} = \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}\big[\|\nabla \mathcal{J}_{\mathrm{BEPA}}(\theta_k)\|_2^2\big], \tag{20}$$

which gives exactly Eq. (13).

Remark (Injected success, group-wise normalization, and experience replay). When a single cached success is injected into $N-1$ failures (Eq. (3)), the reward multiset is $\{1, 0, \ldots, 0\}$ and Eq. (4) produces one positive advantage and $N-1$ negative advantages with $\hat{A}_{\text{succ}} = \sqrt{N-1}$, $\hat{A}_{\text{fail}} = -1/\sqrt{N-1}$, and $\sum_{\tau \in \hat{\mathcal{T}}_x} \hat{A}_\tau = 0$. Thus the group-wise normalization acts as a data-dependent baseline: the injected success and failed rollouts are contrasted within the same group, but the overall scale is bounded by $A_{\max}$ and the second-moment bound Eq. (12) still holds. This design biases each failed group toward the successful trajectory (as intended) while keeping the update within the PPO trust region via the clipping in $w_t(\theta)$.

Moreover, LEVEL-2 refreshes the cache with *the agent's own successful rollouts*, so cached trajectories increasingly coincide with recent on-policy experiences. In this sense, the role of the cache gradually shifts from assimilating external expert traces to providing a bounded-staleness *experience replay* buffer over the agent's own successes (Lin, 1992; Mnih et al., 2015; Schaul et al., 2016). Under the clipped mixed-policy GRPO update, such experience replay is compatible with the variance and convergence guarantees above and connects BEPA to a long line of empirically validated replay-based methods in reinforcement learning.

C Benchmarks and Evaluation Protocols

C.1 OSWorld-Verified

Benchmark. We conduct RLVR training on OSWorld-Verified only, an execution-based real-computer benchmark built upon OSWorld (Xie et al., 2024), consisting of 369 diverse computer-use tasks with per-task setup and evaluation scripts. We summarize dataset statistics in Table 2 and report main results in Table 1 (with per-domain breakdown in Table 7). MMBench-GUI and Online-Mind2web are used as held-out evaluation benchmarks.

Evaluation Protocol. Each OSWorld-Verified task is executed in a fresh virtual-machine snapshot with its official initial-state config; the agent interacts with the desktop by emitting textual UITARS actions, which are mapped to `pyautogui` mouse/keyboard operations in the VM (*cf.* (Xie et al., 2024)). An episode terminates when the agent outputs `finished` or reaches the 15-step limit. After termination, OSWorld runs the example-specific execution-based evaluation script to retrieve relevant files/UI state and returns a binary success signal $R(\tau) \in \{0, 1\}$ (we count a task as solved iff $R(\tau) = 1$). All success rates reported on OSWorld-Verified are the fraction of tasks with $R(\tau) = 1$, averaged over three random seeds.

C.2 MMBench-GUI

Benchmark. MMBench-GUI is a hierarchical, multi-platform benchmark designed to systematically assess GUI agents across four progressive levels of difficulty: GUI Content Understanding (L1), Element Grounding (L2), Task Automation (L3), and Task Collaboration (L4). We evaluate generalization on MMBench-GUI (Wang et al., 2025c), a hierarchical multi-platform GUI benchmark spanning four levels (L1–L4) over multiple platforms, and reporting both success rate (SR) and Efficiency–Quality Area (EQA). We provide benchmark statistics in Table 3 and full results in Tables 8 and 9. Only L1 and L2 results are reported as L3 and L4 task configurations are not publicly available at the time of writing.

Evaluation Protocol. For L1 (GUI Content Understanding), we follow the official protocol of MMBench-GUI (Wang et al., 2025c): the agent receives a single GUI screenshot and a multiple-choice question with options, and must select one option; we compute accuracy (SR) per platform and difficulty, and aggregate scores as a weighted average across platforms as in the original benchmark. For L2 (GUI Element Grounding), the agent is given a screenshot and a textual instruction describing a target element, and must output a click position; a prediction is counted as correct if the point lies inside the annotated bounding box, and we report accuracy over all evaluated elements, averaged across platforms and instruction types (Basic / Advanced) following the MMBench-GUI evaluation metric.

C.3 Online-Mind2Web

Benchmark. Online-Mind2Web (Xue et al., 2025) is a benchmark designed to evaluate the real-world performance of web agents on live websites, featuring 300 tasks across 136 popular sites in diverse domains with reliable LLM-as-a-Judge (WebJudge) automactic evaluation. Based on the number of steps required by human annotators, tasks are divided into three difficulty levels: Easy (1-5 steps, 83 tasks), Medium (6-10 steps, 143 tasks), and Hard (11+ steps, 74 tasks). Dataset statistics are summarized in Table 4, and full results are reported in Table 11. We follow the official Online-Mind2Web evaluation procedure: each task is executed in a real browser session with a fixed step budget and is scored by task success (SR) based on the benchmark's execution-based evaluator.

Evaluation Protocol. We follow the official Online-Mind2Web evaluation setup with WebJudge, an LLM-as-a-judge pipeline. For each task, the agent interacts in a real browser session under a fixed step budget; the task description, full action sequence, and up to 50 screenshots are logged and fed to WebJudge. WebJudge first extracts key requirements from the description (especially comparative terms such as "cheapest" or "most recent"), then filters screenshots by scoring each frame and retaining only informative ones, and finally decides success or failure based on the filtered screenshots, action history, and key requirements. We adopt the benchmark's strict criteria on correct filter application, verifiability from screenshots/actions, and exact satisfaction of range and submission conditions. Following (Xue et al., 2025), we use `o4-mini` as the judge and run evaluation in parallel (up to 60 workers); success rate (SR) is defined as the proportion of tasks judged as SUCCESS by WebJudge.

Domain	# Examples
Chrome	46
GIMP	26
LibreOffice Calc	47
LibreOffice Impress	47
LibreOffice Writer	23
Multi-Apps	101
OS	24
Thunderbird	15
VLC	17
VS Code	23
Total	**369**

Table 2: Statistics of the OSWorld-verified dataset across different domains.

	Windows	MacOS	Linux	iOS	Android	Web	Overall
L1				**L1 - Easy**			
	271	84	196	115	307	221	1194
				L1 - Medium			
	271	84	196	115	307	221	1194
				L1 - Hard			
	271	84	196	115	307	221	1194
L2				**L2 - Basic**			
	271	345	191	314	356	310	1787
				L2 - Advanced			
	272	346	196	330	335	308	1787
L3	145	35	268	-	116	155	719
L4	35	35	101	-	30	47	248
Total	**1536**	**1013**	**1344**	**989**	**1758**	**1483**	**8123**

Table 3: Statistics of the evaluation data in MMBench-GUI.

Task Domain	Distribution (%)
Shopping & E-Commerce	17.6
Entertainment & Media	13.2
Travel & Transportation	11.8
Education	11.0
Technology	8.8
Government & Services	8.1
Health & Medical	6.6
Housing & Real Estate	5.9
Finance & Investment	5.9
Other	5.9
Jobs & Careers	2.9
Food & Recipes	2.2
Total Websites	**136**

Table 4: Statistics distribution of task domains in the Online-Mind2Web dataset.

D Baselines

Baselines on OSWorld-Verified. We compare our approach with a comprehensive suite of competitive baselines on the OSWorld benchmark: (1) Closed-Source LLMs: Claude 3.5 Sonnet, OpenAI o3, Doubao-1.5-Pro (Guo et al., 2025), and others, representing state-of-the-art generalist reasoning capabilities. (2) Open-Source GUI Agents: The UI-TARS family (Qin et al., 2025), ARPO (Lu et al., 2025), OpenCUA series (Wang et al., 2025b), and GUI-Owl-7B (Ye et al., 2025), which are specialized models for GUI interaction. (3) Agent Frameworks: Agent S2 (Agashe et al., 2025), a compositional framework employing a Mixture-of-Grounding technique and Proactive Hierarchical Planning to delegate cognitive tasks; and Jedi (Xie et al., 2025), which utilizes multi-scale models trained on large-scale synthetic grounding data to enhance agentic capabilities. (4) Training Methodologies: We benchmark against various strategies including GRPO (Shao et al., 2024), SFT (training on the converted expert traces, and the conversion prompt is shown in Appendix), RL w/ SFT Loss (using SFT loss during RL training), and SFT+RL (a two-stage training process that continues RL training after SFT). We also compare with LUFFY (Yan et al., 2025), which augments RLVR with off-policy reasoning traces via Mixed-Policy GRPO and policy shaping, and Trace Replacement, which differs from LUFFY by replacing trajectories with off-policy traces only upon group failure and calculating the importance ratio using the old policy likelihood rather than a constant (we build BEPA based on the simple trace replacement). For all trace replacement based methods, we use the converted expert traces $\mathcal{D}_{\text{conv}}$ as the off-policy data.

Baselines on MMBench-GUI. For the MMBench-GUI benchmark, we assess cross-platform generalization using: (1) Proprietary Models: GPT-4o and Claude 3.7; (2) Native GUI Models: Qwen2.5-VL-72B (Bai et al., 2025), Aguvis-72B (Xu et al., 2025), and the UI-TARS series (1.5-7B and 72B-DPO) ; and (3) Modular Agents: Planner-Grounder combinations such as GPT-4o paired with UGround-V1-7B (Qian et al., 2025) or UI-TARS-1.5-7B.

Baselines on Online-Mind2Web. We categorize these baselines into three groups: (1) Proprietary End-to-End Agents, including the industry-leading OpenAI Operator , Google Computer Use, and Claude 3.7 Computer Use, which interact with the browser via raw pixel or accessibility tree inputs; (2) Native GUI Agents, such as UITARS1.5-7B (Qin et al., 2025) and ACT-1 (AI, 2022), which are fine-tuned specifically for grounding and action generation on GUI screenshots; and (3) Modular Agent Frameworks, which decouple planning

and execution, including SeeAct (Zheng et al., 2024), Agent-E (Abuelsaad et al., 2024), Navigator (LaVague) (AI, 2025), and Browser Use (Müller and Žunič, 2024).

E Implementation Details

The learning rate is set to 1×10^{-6}, and training is conducted with a batch size of 16 over 256 parallel virtual environments for 4 epochs. Following DAPO (Yu et al., 2025), we use asymmetric clipping thresholds with $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.3$ to balance exploration and exploitation. We remove the KL divergence regularization term and therefore do not need a reference model during optimization. The optimizer is AdamW (Loshchilov and Hutter, 2019) with a learning rate of 1×10^{-6}, and the training-time sampling temperature is fixed to 1.0.

E.1 SFT on Converted Expert Traces

We train the SFT baseline on the *converted* expert trace set $\mathcal{D}_{\text{conv}}$, consisting of 125 successful OS-World tasks (one converted successful trajectory per task). We then construct step-level instruction–response pairs from the same converted expert traces. Concretely, for each expert task x and each step t in its converted trajectory, we create one training pair $(u_{x,t}, y_{x,t})$, where $u_{x,t}$ is the step input (instruction plus the interaction context at step t) and $y_{x,t}$ is the corresponding target response (the expert step output). This yields 1070 step-level samples in total.

We perform supervised fine-tuning for 3 epochs with the token-level cross-entropy loss. The learning rate is 1×10^{-6} and the batch size is 16.

E.2 RL with SFT Loss (RL+SFT)

During GRPO training, at each iteration we may sample some expert tasks within the RL batch. We then select the SFT samples *only* from those expert tasks appearing in the current RL batch, and compute an auxiliary cross-entropy loss on their corresponding step-level pairs.

Joint objective and weighting. We optimize a combined objective with equal weights for the RL and SFT terms:

$$\mathcal{J}(\theta) = \mathcal{J}_{\text{GRPO}}(\theta) + \lambda \mathcal{L}_{\text{SFT}}(\theta), \qquad (21)$$

with $\lambda = 1$. All GRPO hyperparameters follow Sec. 5.

E.3 LUFFY

Algorithm. LUFFY augments GRPO with *Mixed-Policy GRPO* and *policy shaping*. For each task, a rollout group includes both on-policy trajectories and off-policy expert trajectories, no matter if the rollout traces all fail; advantages are computed by group-wise normalization over the union of rewards from on-/off-policy samples.

Off-policy importance weighting. In principle, the off-policy branch uses an importance ratio $r_t = \pi_\theta(a_t \mid s_t)/\pi_\phi(a_t \mid s_t)$. In practice, LUFFY sets $\pi_\phi = 1$ for off-policy data (i.e., it does not compute teacher token probabilities) to avoid tokenizer/probability incompatibilities and to simplify using existing datasets; correspondingly, the off-policy branch does not apply the PPO-style clipping.

Policy shaping. To mitigate entropy collapse and under-training on low-probability but critical tokens in off-policy traces, LUFFY applies a shaping transform to the off-policy weight: $f(w) = \frac{w}{w+\gamma}$, with $\gamma = 0.1$ by default . Shaping is applied only to the off-policy branch; the on-policy branch follows GRPO.

E.4 Log-Probability on S2 Raw Traces

For the token-probability analysis in Figure 3, we evaluate S2's raw traces under the base policy. S2 is a multi-agent, multi-turn system, but UI-TARS expects a single `system+user` input and a single assistant response. We therefore reconstruct, for each step in an S2 trace, a synthetic single-turn Worker call and compute the log-probability of the corresponding S2 `executor_plan`.

Concretely, each step provides: (i) the global task instruction; (ii) the current subtask and its description; (iii) DAG-derived context (which subtasks are done and which are remaining); (iv) the current reflection (if any); and (v) the current screenshot. We instantiate S2's Worker procedural memory prompt with this context and use the current subtask/reflection plus screenshot as the user message (see Figure 8 and an input/output example in Figure 9). The S2 `executor_plan` at that step is treated as the assistant response.

S2 Worker Prompt Template

System Message (Worker Procedural Memory).
You are an expert in graphical user interfaces and Python code. You are responsible for executing the current subtask: `{current_subtask}` of the larger goal: `{task_instruction}`.
IMPORTANT: The subtasks ['{done_tasks}'] have already been done. The future subtasks ['{future_tasks}'] will be done later. You must **only** perform the current subtask `{current_subtask}` and must not attempt future subtasks.
You are working in {platform}. You are provided with:

- A screenshot of the current time step.
- The history of your previous interactions with the UI (summarized in reflection).
- Access to the following class and methods to interact with the UI:

```
class Agent:
    {agent_api_methods}
```

Your response should be formatted with the following sections:

- **(Previous action verification)**: analyze whether the previous action was successful.
- **(Screenshot Analysis)**: describe the current state of the desktop and open applications.
- **(Next Action)**: decide on the next action in natural language.
- **(Grounded Action)**: output a single Python call using the Agent API:

````
```python
agent.xxx(...)
```
````

User Message (per step).
For the first step of a subtask:

```
Remember only complete the subtask: {current_subtask}.
You can use this extra information for the current subtask:
{current_subtask_info}.
Text Buffer = [].
```

For subsequent steps (with reflection):

```
You may use this reflection on the previous action and trajectory:
{reflection_text}
Text Buffer = [].
```

In both cases, the user message is paired with the current screenshot as the vision input.

Figure 8: S2 Worker prompt template used to reconstruct single-turn inputs for log-probability evaluation of S2 raw traces under UI-TARS1.5-7B.

Example Input/Output for Raw S2 Trace (Step 1)

System Message.
You are an expert in graphical user interfaces and Python code. You are responsible for executing the current subtask: ‘Focus Chrome window‘ of the larger goal: ‘Can you make my computer bring back the last tab I shut down¿. IMPORTANT: The subtasks [”] have already been done. The future subtasks [’Restore last closed tab’] will be done in the future by me. You must only perform the current subtask: ‘Focus Chrome window‘. Do not try to do future subtasks.
You are working in linux. You must only complete the subtask provided and not the larger goal.
You are provided with:

- A screenshot of the current time step.
- The history of your previous interactions with the UI.
- Access to an Agent API to interact with the UI.

Your response should be formatted with the following sections:

- **(Previous action verification)**: analyze whether the previous action was successful.
- **(Screenshot Analysis)**: describe the current state of the desktop and open applications.
- **(Next Action)**: decide on the next action in natural language.
- **(Grounded Action)**: output a single Python call using the Agent API, e.g.

````
```python
agent.click("...", 1, "left")
```
````

User Message (instance-specific).

```
Remember only complete the subtask: Focus Chrome window
You can use this extra information for completing the current subtask:
- Click an empty spot inside the Chrome window
  - Purpose: make Chrome the active application.
Text Buffer = [].
```

Screenshot.

- Input image: step 0’s screenshot

Ground-Truth Assistant Response (executor_plan).

````
(Previous action verification)
No previous action has been taken yet, so nothing to verify.

(Screenshot Analysis)
The desktop shows an Ubuntu environment. A Google Chrome window is
present in the center displaying the Airbnb webpage; it appears to be
the active application (the top bar reads "Google Chrome"). To ensure it
is definitely in focus for later steps, we should click anywhere within
the webpage that doesn't interact with page elements.

(Next Action)
Click an empty spot inside the current Chrome window (e.g., a blank
portion of the Airbnb page) to guarantee Chrome is focused.

(Grounded Action)
```python
agent.click("A blank area in the middle of the Airbnb webpage within the Google
    Chrome window, away from any buttons or links", 1, "left")
```
````

Figure 9: Example single-turn input/output used to compute token-level log-probabilities for an S2 Worker step under UI-TARS1.5-7B (Step 1 of the “Focus Chrome window” subtask).

E.5 Codebase: Verl-GUI

As part of the new assets released with this work, we propose `Verl-GUI`, a highly scalable distributed training framework for long-horizon, multi-turn vision-language GUI agent training. Our `Verl-GUI` is built upon the veRL framework (Sheng et al., 2025) and extends it with several features to enable scalable reinforcement learning for agentic tasks involving complex computer environments. Verl-GUI supports both synchronous (default) and asynchronous training modes, comprising approximately 212K lines of code across 714 Python files, representing an 83% increase over the original veRL framework (116K lines, 543 files). Key capabilities of our framework include:

Heterogeneous Cluster Architecture. Verl-GUI completely separates trainer and rollout into independent Ray clusters in both sync and async modes, enabling deployment across heterogeneous compute resources. Nodes with high-bandwidth interconnects (InfiniBand, NVLink) are allocated for training where gradient synchronization demands intensive inter-GPU communication, while nodes with standard PCIe connectivity suffice for rollout where environment interactions are largely independent.

Multiple Storage Backends. The framework supports Azure Blob Storage, NAS, and local filesystems through a unified abstraction layer, enabling seamless deployment across cloud and on-premise infrastructure.

Async Task Queue. In asynchronous mode, Verl-GUI dynamically maintains a task queue for the rollout cluster to consume, enabling decoupled and non-blocking task processing where rollout and training proceed independently.

K-round Rollout Processing. The system intelligently splits batches across multiple rounds when the trainer's global batch size exceeds rollout cluster capacity. For example, if the trainer requires 128 samples but the rollout cluster can only process 64 concurrently, the system automatically executes $K = 2$ rounds, enabling flexible scaling regardless of cluster size mismatches.

Scalable Parallel Environment Execution. The number of concurrent environments scales with rollout cluster compute capacity (i.e., max_envs = num_gpus $\times$ batch_per_gpu), with Ray-based orchestration and automatic Docker cleanup.

Service-oriented Rollout Orchestration. Verl-GUI employs modular components including CheckpointManager, EnvWorkerPool, RolloutService, and ValidationAdapter for clean separation of concerns.

F Interface and Prompt Templates

F.1 UI-TARS Action Space

We adopt the UI-TARS action interface throughout training and evaluation. Table 5 lists the complete action space and the parameterization used in our implementation.

F.2 Agent S2 Action Space

We provide the expert's action space. Table 6 lists the full action space together with natural-language descriptions and argument specifications.

F.3 Prompt Templates

We use a unified prompt format across all methods unless otherwise stated.

System prompt. The system prompt that constrains the model to output `Thought` and `Action` is shown in Figure 11.

Plan summarization prompt. To abstract an expert trajectory into a compact natural-language plan $p_x = \phi(\tau_x^E)$ for LEVEL-1 self-rolling (Sec. 4.1), we use the plan summarization template in Figure 13.

Trace conversion prompt. To convert framework-produced expert traces into the end-to-end policy format (i.e., instruction–response pairs compatible with UI-TARS), we use the conversion prompt in Figure 12. Converted traces form $\mathcal{D}_{\text{conv}}$ for SFT/RL+SFT baselines (Appendix E.1–E.2), and also serve as the expert source $\mathcal{D}_E$ used in BEPA initialization (Sec. 4.1).

Environment	Action	Definition
Shared	Click(x, y)	Clicks at coordinates (x, y).
	Drag(x1, y1, x2, y2)	Drags from (x1, y1) to (x2, y2).
	Scroll(x, y, direction)	Scrolls at (x, y) in the given direction.
	Type(content)	Types the specified content.
	Wait()	Pauses for a brief moment.
	Finished()	Marks the task as complete.
	CallUser()	Requests user intervention.
Desktop	Hotkey(key)	Presses the specified hotkey.
	LeftDouble(x, y)	Double-clicks at (x, y).
	RightSingle(x, y)	Right-clicks at (x, y).
Mobile	LongPress(x, y)	Long presses at (x, y).
	PressBack()	Presses the "back" button.
	PressHome()	Presses the "home" button.
	PressEnter()	Presses the "enter" key.

Table 5: The UITARS action space for different platforms. For OSWorld, the agent typically uses shared and desktop actions.

Agent Action	Description	Arguments
click	Click on an element.	`element_description`, `num_clicks`, `button_type`, `hold_keys`
type	Type text into an element.	`element_description`, `text`, `overwrite`, `enter`
scroll	Scroll within an element.	`element_description`, `clicks`, `shift`
hotkey	Press a hotkey combo.	`keys`
hold_and_press	Hold keys and press others.	`hold_keys`, `press_keys`
drag_and_drop	Drag and drop between elements.	`element_description_1`, `element_description_2`, `hold_keys`
save_to_knowledge	Save data to a per-task memory.	`text`
switch_applications	Switch to another app.	`app_name`
highlight_text_span	Highlight a text span.	`starting_phrase`, `ending_phrase`
set_cell_values	Set tabular cell values.	`cell_values`, `app_name`, `sheet_name`
wait	Wait for some time.	`time`
done	Mark subtask as success.	None
fail	Mark subtask as failure.	None

Table 6: **Agent S2 action space, descriptions, and arguments.**

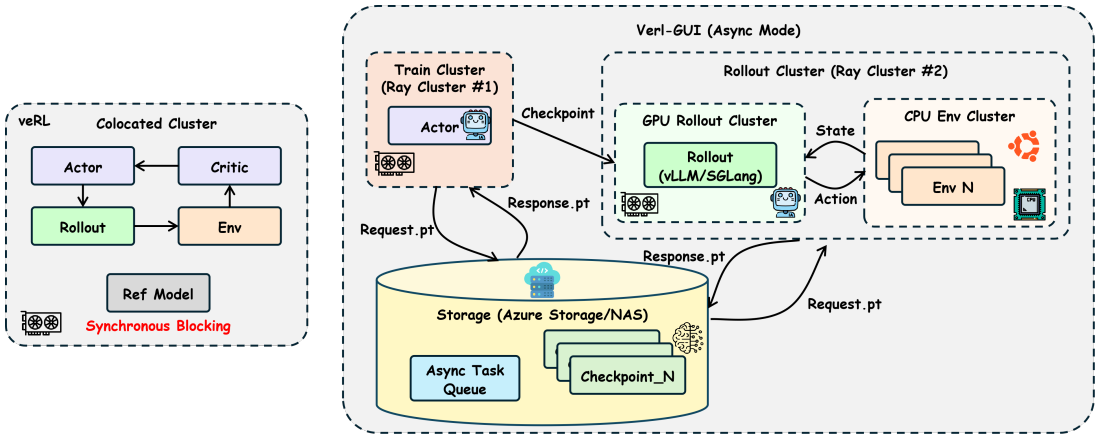


Figure 10: **GUI agent training framework comparison. Left:** veRL uses a single colocated cluster where all workers (Actor, Critic, Rollout, Env) share resources and communicate synchronously, causing blocking during environment interactions. **Right:** Verl-GUI separates training and rollout into independent Ray clusters in both sync and async modes, enabling heterogeneous hardware allocation (IB/NVLink nodes for training, PCIe nodes for rollout). The rollout cluster further divides into GPU sub-cluster (for model inference) and CPU sub-cluster (for Docker-based environment execution). In async mode, an async task queue dynamically maintains tasks for decoupled consumption.

System Prompt Template for UITARS

You are a GUI agent. You are given a task and your action history, with screenshots. You need to perform the next action to complete the task.

Output Format

```
Thought: ...
Action: ...
```

Action Space

```
click(start_box='<|box_start|>(x1,y1)<|box_end|>')
left_double(start_box='<|box_start|>(x1,y1)<|box_end|>')
right_single(start_box='<|box_start|>(x1,y1)<|box_end|>')
drag(start_box='<|box_start|>(x1,y1)<|box_end|>',
     end_box='<|box_start|>(x3,y3)<|box_end|>')
hotkey(key='')
type(content='xxx')  # Use \', \" and \n in content
scroll(start_box='<|box_start|>(x1,y1)<|box_end|>',
       direction='down or up or right or left')
wait()
finished(content='xxx')
```

Note

- Use English in both the `Thought` and `Action` parts.
- In the `Thought` part, write a brief plan and end with one sentence that clearly summarizes your next action and its target element.

User Instruction

```
{instruction}
```

Figure 11: The system prompt used to instruct the GUI agent to produce thought and action.

Prompt Template for S2→UI-TARS Trace Conversion

You are an expert at converting S2 GUI automation traces into UI-TARS format.

Your Task: Transform S2's `{executor_plan}`, `{reflection}`, and `{pyautogui_action}` into UI-TARS format with both natural language `Thought` and action notation.

Input (S2 Format):

- `executor_plan`: screenshot analysis and next-action planning.
- `reflection`: reflection on current progress (may be `null`).
- `pyautogui_action`: the executed PyAutoGUI code (e.g., `pyautogui.click(1265, 245, clicks=1, button='left')`).

Output (UI-TARS Format):

```
Thought: {{natural language reasoning in 1-3 sentences}}
Action: {{UI-TARS action notation}}
```

UI-TARS Action Space: you must output actions using the following templates:

```
click(start_box='<|box_start|>(x1,y1)<|box_end|>')
left_double(start_box='<|box_start|>(x1,y1)<|box_end|>')
right_single(start_box='<|box_start|>(x1,y1)<|box_end|>')
drag(start_box='<|box_start|>(x1,y1)<|box_end|>',
     end_box='<|box_start|>(x3,y3)<|box_end|>')
hotkey(key='key1 key2 ...')
type(content='xxx')  # Use \', \", and \n with escaping.
scroll(start_box='<|box_start|>(x1,y1)<|box_end|>',
       direction='down or up or right or left')
wait()  # Sleep for 5s and take a screenshot
finished(content='xxx')
```

Critical Requirements:

1. **Box tokens:** you *must* include `<|box_start|>` and `<|box_end|>` around coordinates in `start_box` and `end_box`.
2. **Coordinate format:** `<|box_start|>(x,y)<|box_end|>` where `x,y` are numbers.
3. **Hotkey:** space-separated keys (e.g., `'ctrl c'`, `'ctrl shift b'`, or a single key like `'pagedown'`).
4. **Scroll:** `start_box` is optional—include it only if the PyAutoGUI action specifies coordinates.
5. **Escaping:** use `\'` for single quotes, `\"` for double quotes, and `\n` for newlines.

Thought Guidelines:

1. Use first-person perspective (*"I see...", "I notice...", "Let me..."*).
2. Be natural and conversational, not robotic.
3. Keep it concise.
4. Focus on what you observe and what you plan to do next.

Example (simplified): Given `{executor_plan}`, `{reflection}`, and `{pyautogui_action}`, output:

```
Thought: I see the Chrome window open on the Puzzle Game 2048 page.
        To access more options, I should open the browser menu.
Action: click(start_box='<|box_start|>(1265,245)<|box_end|>')
```

Figure 12: Prompt for converting Agent S2's traces into UI-TARS-style thoughts and actions.

Prompt Template for Plan Summary

You are an expert at converting technical GUI automation traces into clear, actionable step-by-step plans.

Your Task: Analyze the complete execution trace and generate a clean, effective step-by-step plan. The trace contains *all* actions that were performed, including exploratory actions, mistakes, and corrections. Your job is to extract only the *effective* actions that contributed to task completion.

Critical Requirements:

1. **Filter out exploratory / trial-and-error steps:** remove actions that
   - were exploratory attempts that did not work,
   - were mistakes that needed correction,
   - were redundant or duplicated,
   - were verification steps that did not contribute to progress.
2. **Keep only effective steps:** include actions that
   - successfully moved the task forward,
   - were necessary for completion,
   - represent the optimal path to achieve the goal.
3. **Consolidate related actions:** if multiple execution steps accomplish one logical action, combine them into a single step.

Output Format: Generate a clean, numbered step-by-step plan that:

1. starts with a brief task description,
2. lists each action as a simple, clear step,
3. uses action verbs (click, type, select, drag, press, etc.),
4. removes technical details and purposes,
5. combines related sub-steps when logical,
6. maintains the essential flow of actions,
7. contains *only* effective actions (no trial-and-error or exploration).

Example:

```
Task: Create a desktop shortcut for the current website

1. Press Ctrl+L to focus the address bar
2. Type "mathsisfun.com/games/2048.html" and press Enter
3. Click the three-dot Chrome menu button (top-right)
4. Click "Cast, save and share"
5. Click "Create shortcut..."
6. Click the "Create" button in the dialog
```

Now convert the following trace into a step-by-step plan:

Figure 13: Prompt for converting S2 execution traces into a concise step-by-step plan.

G Full Benchmark Results

We reports full quantitative results for each benchmark used in our study for three benchmarks. Table 7 reports per-domain success rates (%) on OSWorld-Verified for all compared methods. For MMBench-GUI, we report detailed results for each level in Tables 8 and 9. Table 11 reports success rates (SR %) for all methods evaluated on Online-Mind2Web.

H Extensibility

Setup. To evaluate how BEPA generalizes across expert sources and base models, we instantiate BEPA with different combinations of experts and backbones on OSWorld-Verified. On UITARS1.5-7B, we use Agent S2, GUI-Owl-7B, or a hybrid configuration (S2+Owl), where for each task we *prefer* the self-rolled success from Agent S2 when available and fall back to the GUI-Owl trace otherwise (i.e., S2 takes precedence on tasks where both experts succeed). For GUI-Owl-7B's step by step plan extraction, we use the same plan summarization prompt as in Figure 13. On OpenCUA-7B, we additionally train BEPA with Agent S2 as the expert (Table 10).

Expert diversity on UITARS1.5-7B. With Agent S2 as the expert, BEPA reaches 32.40% overall success, improving over GRPO (UITARS) at 24.50%. Swapping the expert to GUI-Owl-7B still yields a strong 31.55% overall success, with notable gains on Writer and TB (Thunderbird), indicating that BEPA can effectively exploit a different GUI-specialized expert. Combining Agent S2 and GUI-Owl (S2+Owl) further boosts performance to 34.71% overall, capturing complementary strengths: S2 remains strong on Chrome, Impress, OS, and VSCode, while Owl provides substantial gains on Writer, Multi-app, TB, and VLC.

Portability across base policies. BEPA also transfers to a different base model. On OpenCUA-7B, BEPA with Agent S2 as the expert attains 31.98% overall success, a large improvement over the OpenCUA baseline at 24.30% and comparable to BEPA on UITARS1.5-7B. Across both backbones, BEPA consistently turns stronger experts into tangible, per-domain gains on OSWorld-Verified, demonstrating that our bi-level assimilation mechanism is not tied to a specific base model or a single expert source.

I Sensitivity Analysis

We study the robustness of BEPA to two key hyperparameters: (i) the rollout group size N used in GRPO, and (ii) the LEVEL-2 cache update rule $\mathcal{U}$. We report the average success rate and standard deviation across 3 runs for all experiments.

Rollout Group Size N. The rollout group size N controls the exploration width per task before group-wise normalization. Table 12(a) demonstrates that increasing N from 4 to 16 monotonically improves the average success rate on OSWorld-Verified ($29.85_{\pm 1.12}\% \rightarrow 33.40_{\pm 0.58}\%$). Larger groups increase the probability of discovering rare successful trajectories on hard tasks, which subsequently triggers more frequent LEVEL-2 cache updates (as seen in Figure 14). Although $N{=}16$ yields the highest performance, we adopt $N{=}8$ ($32.13_{\pm 0.25}\%$) as the default setting to balance computational cost and final performance.

Cache Update Rule $\mathcal{U}$. We compare three LEVEL-2 cache update rules $\mathcal{U}(\mathcal{T}_x^{\text{succ}})$ for selecting a new cached trajectory: *Random* (default), *Highest LogProb*, and *Shortest Step*. *Highest LogProb* prioritizes trajectories with higher likelihood under the current policy, a heuristic often linked to better quality in self-evolving LLMs (Huang et al., 2023, 2025), while *Shortest Step* favors execution efficiency from the GUI agent's perspective. As shown in Table 12(b), while both heuristic strategies achieve slightly higher average success rates—*Highest LogProb* at $33.12_{\pm 0.45}\%$ and *Shortest Step* at $33.55_{\pm 0.62}\%$—compared to *Random* ($32.13_{\pm 0.25}\%$), these gains are marginal and not statistically significant given the overlapping standard deviations. This implies that the primary driver of BEPA's success is the **mechanism of dynamic updating** itself—which progressively aligns off-policy guidance with the on-policy manifold—rather than the specific criterion used to filter successes. Since any successful rollout from the current policy already provides valid on-manifold guidance, we retain **Random** as the default $\mathcal{U}$ for its simplicity and robustness.

J Domain Breakdown of BEPA's Gains

To understand where BEPA improves over the base UITARS1.5-7B policy and the Agent S2 expert, we decompose OSWorld-Verified successes by their source (Figure 15). For each domain, we count tasks that BEPA (i) shares with UITARS1.5-7B (re-

Method	Chrome	GIMP	Calc	Impress	Writer	Multi	OS	Thunderbird	VLC	VSCode	Overall
Agentic Framework											
Agent S2	29.06	42.69	13.16	27.38	36.69	**23.62**	**67.28**	64.87	42.41	52.65	33.00
Jedi-7B w/ o3	**52.09**	**65.38**	**29.79**	**42.53**	**65.22**	20.43	50.00	**73.33**	**47.06**	56.52	**42.40**
General Model											
OpenAI o3	6.52	11.54	0.00	0.00	4.35	11.83	37.50	6.67	10.88	13.04	9.10
Doubao-1.5-Think-Vision	47.83	50.00	25.53	36.15	43.48	6.45	33.33	66.67	35.29	47.83	**31.90**
Claude-4-Sonnet	36.87	46.15	17.02	36.17	43.48	9.68	37.50	66.67	38.47	60.87	31.20
Specialized Model											
UITARS-72B-DPO	37.60	61.54	8.70	25.53	30.43	4.92	33.33	40.00	17.65	39.13	24.00
Computer-Use-Preview	36.87	34.62	10.64	25.45	30.43	10.75	45.83	46.67	29.41	47.83	26.00
GUI-Owl-7B	41.22	**65.38**	17.02	19.06	**52.17**	9.68	**50.00**	**66.67**	29.41	**65.22**	**32.10**
OpenCUA-7B	36.87	50.00	10.64	36.15	26.09	6.52	30.43	53.33	29.41	43.48	24.30
ARPO	22.02	50.00	12.77	19.57	39.12	8.11	37.50	40.00	23.53	39.13	22.97
UITARS1.5-7B	22.73	50.00	8.51	25.33	39.12	7.08	34.78	33.33	23.53	56.52	23.66
UITARS1.5-7B+											
SFT	29.75	37.84	10.46	14.66	25.65	4.23	25.66	19.68	11.58	42.77	18.00
GRPO	28.15	53.18	15.84	20.81	36.98	9.75	27.74	21.27	25.72	60.11	24.50
RL+SFT	26.63	46.23	14.80	19.62	30.36	8.22	26.15	20.08	24.29	52.31	22.00
SFT+RL	21.63	30.94	4.70	7.12	19.22	4.74	41.44	14.74	33.22	24.03	15.20
Trace Replacement	29.16	58.35	12.90	15.07	35.17	10.38	33.71	40.45	23.80	67.77	25.50
LUFFY	28.93	36.16	11.43	25.72	40.86	5.78	51.82	35.83	34.51	70.08	25.80
BEPA (ours)	**47.80**	**69.20**	10.64	**31.90**	34.80	8.60	**54.20**	40.00	**29.41**	**73.90**	**32.40**

Table 7: **Performance comparison on OSWorld-Verified.** We report the best single-run success rates (%) across 10 application domains. Methods are grouped into four categories: agentic frameworks with compositional architectures, general-purpose LLMs, specialized GUI models, and UITARS1.5-7B with various training strategies. BEPA achieves the highest overall performance among all UITARS1.5-7B+ variants, demonstrating the effectiveness of bi-level expert-to-policy assimilation.

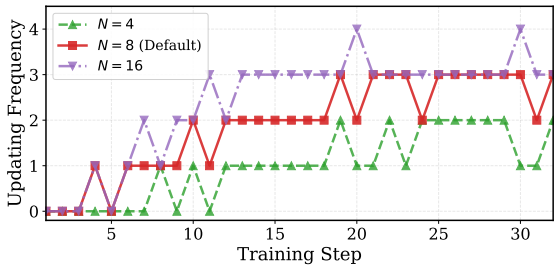


Figure 14: **Effect of rollout group size N on LEVEL-2 updates.** We plot the per-step cache-updating frequency for different group sizes (N=4, 8, 16). Larger N leads to more frequent cache refreshes, especially in later stages of training, indicating faster discovery of successful rollouts and more aggressive expert-to-policy assimilation.

tained), (ii) newly solves that are also solved by Agent S2 but not UITARS1.5-7B (assimilated), and (iii) solves while both baselines fail (self-evolved). BEPA closes much of the gap to Agent S2 on several domains: for example, on Chrome it solves 22 tasks versus 13 for Agent S2 and 10 for UITARS1.5-7B, combining strong retention with substantial assimilation and a non-trivial number of self-evolved successes. In contrast, multi-application workflows remain challenging: BEPA reaches only 8 successes where Agent S2 solves 22, suggesting that when both the base policy and expert traces are far from the target distribution, bi-level assimilation alone is insufficient.

K Case Studies

Self-rolling as Learnable Guidance. Figure 16 compares Agent S2, a converted trace, and a self-rolled trace on the Thunderbird task "Attach my test file to the email and send." All three complete the task in five steps, but the converted trace directly copies three expert hotkeys (Ctrl+Shift+A, Ctrl+L, Enter) together with S2-style template thoughts (e.g., "I see ...", "I notice ...", "The task has been completed successfully as per the plan."), where the "plan" refers to S2's global plan rather than UITARS1.5-7B's own reasoning. This produces actions and thoughts that are valid but lie off the policy's natural distribution. In contrast, the self-rolled trace only conditions on a short expert plan (click Attach, go to Home, select `aws-bill.pdf`, click Open) and lets the policy choose its own clicks and narrative. The resulting trajectory follows a menu-based path with concise, observation-grounded thoughts, illustrating how plan-conditioned self-rolling rewrites expert guidance into a form that is easier for the end-to-end policy to absorb.

Model	Windows	macOS	Linux	iOS	Android	Web	Overall
Easy Level							
GPT-4o	62.47	62.38	67.89	58.52	56.41	58.51	60.16
Claude-3.5	41.34	41.61	50.04	42.03	38.96	41.79	41.54
Claude-3.7	34.66	39.37	49.05	42.76	37.45	40.80	39.08
Qwen-Max-VL	69.05	69.91	72.51	70.82	63.09	69.46	68.15
Qwen2.5-VL-72B	65.86	73.02	75.23	67.24	58.09	72.08	66.98
UI-TARS-72B-DPO	41.59	35.16	28.52	31.08	52.25	35.33	40.18
InternVL3-72B	74.67	79.16	78.72	**83.57**	80.10	81.18	79.15
LUFFY	84.19	82.38	88.01	81.37	83.94	88.67	85.18
UI-TARS1.5-7B	83.82	82.28	84.03	81.22	83.64	85.12	83.69
GRPO	84.44	82.63	88.26	81.62	84.19	88.92	85.43
BEPA (Ours)	**88.94**	**87.13**	**94.26**	**86.12**	**88.69**	**94.92**	**90.46**
Medium Level							
GPT-4o	56.33	59.70	63.13	54.06	57.69	54.98	57.24
Claude-3.5	39.28	45.97	47.63	44.57	42.03	34.33	41.26
Claude-3.7	39.34	42.28	39.23	39.45	36.05	36.17	38.39
Qwen-Max-VL	63.40	66.90	73.85	68.02	63.66	64.59	65.44
Qwen2.5-VL-72B	66.29	72.73	72.63	59.27	66.24	68.24	67.45
UI-TARS-72B-DPO	38.83	37.14	41.60	41.72	54.74	31.55	41.77
InternVL3-72B	71.46	79.88	78.58	78.43	81.36	78.67	77.89
LUFFY	95.33	83.97	89.88	80.67	89.09	83.89	88.50
UI-TARS1.5-7B	95.16	84.07	86.90	80.72	88.99	81.34	87.49
GRPO	95.58	84.22	90.13	80.92	89.34	84.14	88.75
BEPA (Ours)	**98.28**	**86.92**	**94.13**	**83.62**	**92.04**	**88.14**	**91.91**
Hard Level							
GPT-4o	60.69	60.38	52.42	45.27	50.93	50.83	53.49
Claude-3.5	37.40	42.70	34.07	40.86	36.96	38.11	37.55
Claude-3.7	32.99	34.48	31.97	39.20	36.96	38.92	35.65
Qwen-Max-VL	66.64	67.59	65.80	60.23	58.78	65.34	63.69
Qwen2.5-VL-72B	70.68	68.91	70.98	57.59	53.94	68.10	64.56
UI-TARS-72B-DPO	31.48	35.87	24.19	36.33	58.13	19.94	35.78
InternVL3-72B	75.08	76.19	77.44	70.37	75.73	78.11	75.70
LUFFY	90.00	90.24	96.91	80.72	89.78	93.22	90.80
UI-TARS1.5-7B	90.03	90.54	95.43	80.97	89.88	92.17	90.44
GRPO	90.25	90.49	97.16	80.97	90.03	93.47	91.05
BEPA (Ours)	**91.65**	**91.89**	**98.80**	**82.37**	**91.43**	**95.47**	**92.60**

Table 8: **Performance on L1-GUI Content Understanding.** We report the success rates (%) (SR) of various models across all platforms and the weighted average success rate.

Assimilation from Off-policy Traces. Figure 17 shows how BEPA acquires a new capability from off-policy guidance on the OS task "Remove Vim from the favorites list." UITARS1.5-7B fails to interpret "favorites" correctly and never manipulates the dock, whereas the Agent S2 expert solves the task by right-clicking the Vim icon in the dock and selecting "Remove from Favorites." After training with BEPA, the end-to-end policy solves the same task by following a similar high-level strategy, even though it could not solve it before. This example illustrates LEVEL-2's role in turning successful expert trajectories into lasting new skills beyond the base policy.

Model	Windows		MacOS		Linux		iOS		Android		Web		Avg.	
	Basic	Adv.	Basic	Adv.	Basic	Adv.	Basic	Adv.	Basic	Adv.	Basic	Adv.	Basic	Adv.
GPT-4o	1.48	1.10	8.69	4.34	1.05	1.02	5.10	3.33	2.53	1.41	3.23	2.92	3.68	2.35
Claude-3.7	1.48	0.74	12.46	7.51	1.05	0.00	13.69	10.61	1.40	1.40	3.23	2.27	5.55	3.75
Qwen-Max-VL	43.91	36.76	58.84	56.07	53.93	30.10	77.39	59.09	79.49	70.14	74.84	58.77	64.73	51.82
Aguvis-7B-720P	37.27	21.69	48.12	33.27	33.51	25.00	67.52	65.15	60.96	50.99	61.61	45.45	51.50	40.26
ShowUI-2B	9.23	4.41	24.06	10.40	25.13	11.73	28.98	19.70	17.42	8.73	22.90	12.66	21.29	11.27
OS-Atlas-Base-7B	36.90	18.75	44.35	21.68	31.41	13.27	74.84	48.79	69.60	46.76	61.29	35.39	53.06	30.77
UGround-V1-7B	66.79	38.97	71.30	48.55	56.54	31.12	92.68	70.91	**93.54**	70.99	88.71	64.61	78.26	54.19
InternVL3-72B	70.11	42.64	75.65	52.31	59.16	41.33	93.63	80.61	92.70	78.59	90.65	65.91	80.32	60.23
Qwen2.5-VL-72B	55.72	33.82	49.86	30.06	40.31	20.92	56.05	28.18	55.62	25.35	68.39	45.78	54.33	30.69
Qwen2.5-VL-7B	31.37	16.54	31.30	21.97	21.47	12.24	66.56	55.15	35.11	35.21	40.32	32.47	37.69	28.93
UI-TARS1.5-7B	76.75	45.59	77.39	52.89	67.02	40.31	92.04	70.00	93.26	71.83	87.74	61.36	83.72	58.57
UI-TARS-72B-DPO	78.60	**51.84**	80.29	**62.72**	68.59	**51.53**	90.76	**81.21**	92.98	**80.00**	88.06	**68.51**	84.50	**67.49**
LUFFY	76.95	45.49	77.59	52.79	71.72	43.01	92.24	69.90	93.46	71.73	92.44	64.06	85.18	59.26
GRPO	77.25	45.79	77.89	53.09	72.02	43.31	92.54	70.20	93.76	72.03	92.74	64.36	85.48	59.56
BEPA (Ours)	**80.25**	47.59	**80.89**	54.89	**76.02**	45.81	**95.54**	72.00	**96.76**	73.83	**96.74**	66.86	**88.76**	61.56

Table 9: **Performance on the L2-GUI Element Grounding.** 'Adv.' stands for advanced. 'Avg.' reports the weighted average score across platforms for each mode (Basic/Adv.).

Method	**Chrome**	**GIMP**	**Calc**	**Impress**	**Writer**	**Multi**	**OS**	**TB**	**VLC**	**VSCode**	**Overall**
Base Models & Baselines											
UITARS-1.5-7B	22.73	50.00	8.51	25.33	39.12	7.08	34.78	33.33	23.53	56.52	23.66
OpenCUA-7B	36.87	50.00	10.64	26.09	36.15	6.52	30.43	53.33	29.41	43.48	24.30
GRPO (UITARS)	28.15	53.18	15.84	20.81	36.98	9.75	27.74	21.27	25.72	60.11	24.50
BEPA Variants (UITARS-1.5-7B)											
BEPA (w/ Agent S2)	**47.80**	**69.20**	10.64	**34.80**	31.90	8.60	**54.20**	40.00	29.41	**73.90**	32.40
BEPA (w/ GUI-Owl)	36.96	61.54	**12.77**	25.53	**52.17**	**12.80**	41.67	60.00	35.29	60.87	31.55
BEPA (S2 + Owl)	46.50	68.00	12.50	33.50	50.00	9.95	52.00	58.00	**36.00**	72.00	**34.71**
BEPA Variants (OpenCUA-7B)											
BEPA (OpenCUA + S2)	43.48	57.69	14.89	31.91	43.48	10.78	45.83	**66.67**	32.00	52.17	31.98

Table 10: **Extensibility.** We report the success rates (%) on OSWorld-Verified. The results show BEPA's effectiveness across different base models (UITARS, OpenCUA) and its ability to scale with hybrid expert guidance (S2 + Owl).

Agent	Easy	Medium	Hard	Average SR
Navigator	90.1	76.2	71.1	78.7
Google Computer Use	77.1	71.3	55.4	69.0
Operator	83.1	58.0	43.2	61.3
ACT-1 (2025-08-23)	81.9	54.5	35.1	57.3
Claude Computer Use 3.7	90.4	49.0	32.4	56.3
ACT-1 (2025-07-16)	65.1	46.2	23.0	45.7
SeeAct	60.2	25.2	8.1	30.7
Browser Use	55.4	26.6	8.1	30.0
Claude Computer Use 3.5	56.6	20.3	14.9	29.0
Agent-E	49.4	26.6	6.8	28.0
UITARS1.5-7B	46.91	18.88	5.26	23.28
LUFFY	48.91	20.68	5.66	24.79
GRPO	49.41	20.38	5.76	24.81
BEPA (Ours)	**53.42**	**24.17**	**8.92**	**28.52**

Table 11: **Performance comparison on Online-Mind2Web.** We report the success rates (%) (SR) of various agents across three difficulty levels and the weighted average success rate. BEPA demonstrates superior generalization on unseen websites compared to baselines, achieving a substantial gain over the base model (+5.24%) and outperforming several modular frameworks.

Group Size	**Success Rate (%)**
$N = 4$	$29.85_{\pm 1.12}$
$N = 8$ (Default)	$32.13_{\pm 0.25}$
$N = 16$	$\mathbf{33.40}_{\pm 0.58}$

(a) Rollout group size N.

Update Rule $\mathcal{U}$	**Success Rate (%)**
Shortest Step	$\mathbf{33.55}_{\pm 0.62}$
Highest LogProb	$33.12_{\pm 0.45}$
Random (Default)	$32.13_{\pm 0.25}$

(b) Cache update rule $\mathcal{U}$.

Table 12: **Sensitivity on OSWorld-Verified.** Average overall success rate with standard deviation across 3 runs under different rollout group sizes N (left) and LEVEL-2 cache update rules $\mathcal{U}$ (right).

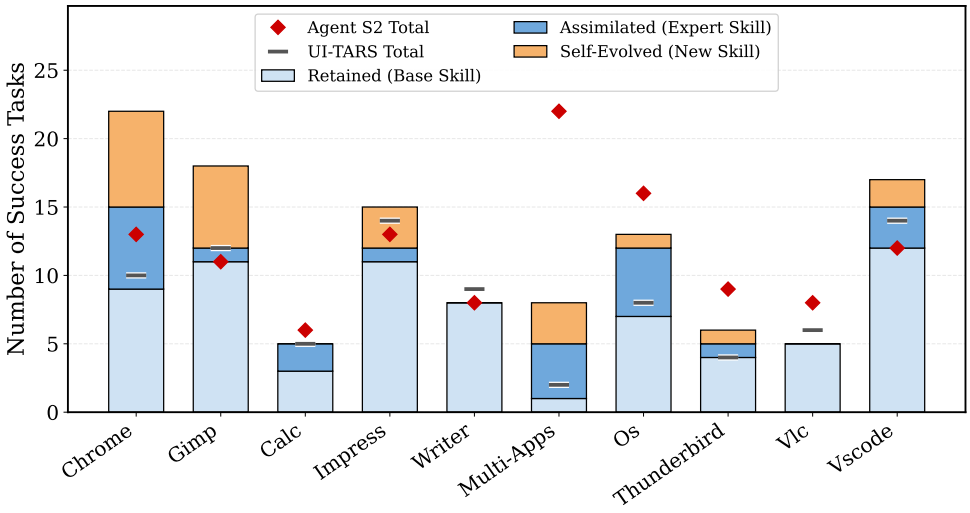
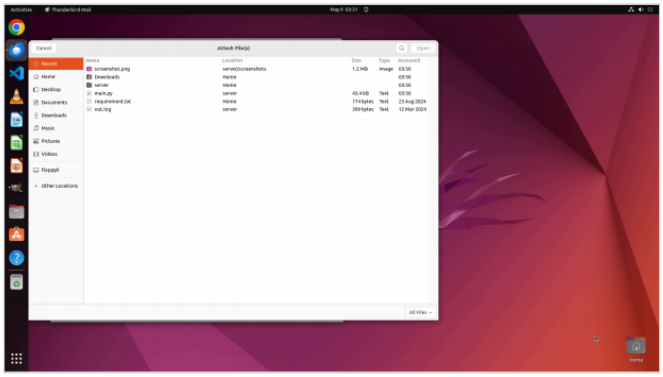


Figure 15: **Domain breakdown of BEPA's gains.** For each domain, BEPA successes are decomposed into retained tasks (also solved by UITARS1.5-7B), assimilated tasks (solved by Agent S2 but not by UITARS1.5-7B), and self-evolved tasks (solved only by BEPA). Red diamonds and gray ticks denote the total successes of Agent S2 and UITARS1.5-7B, respectively.

Agent S2 (Expert Framework)

5 steps → Success



Step 1 Expert hotkey

Goal Plan:

1. **Open attachment dialog**:
- Press Ctrl+Shift+A
- Purpose: Launch file chooser

2. **Focus location bar**:
- Press Ctrl+L
- Purpose: Enable direct path entry

3. **Select bill file**:
- Type ~/aws-bill.pdf and press Enter
- Purpose: Highlight the PDF

4. **Attach file**:
- Press Enter
- Purpose: Add aws-bill.pdf to draft

5. **Keep draft open**:
- Do nothing further
- Purpose: Leave email unsent

Current Subtask:

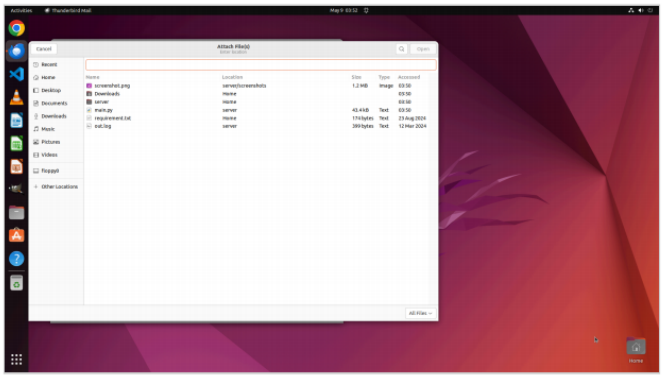
Open attachment dialog

Action:

hotkey('ctrl, shift, a')

Code:

```
import pyautogui; pyautogui.hotkey('ctr
l', 'shift', 'a')
```



Step 2 Expert hotkey

Current Subtask:

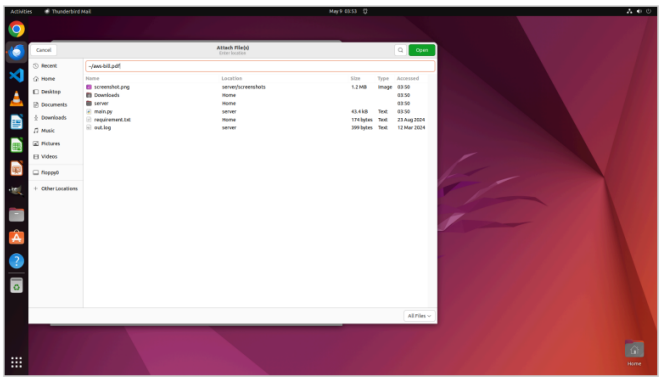
Focus location bar

Action:

hotkey('ctrl, l')

Code:

```
import pyautogui; pyautogui.hotkey('ctr
l', 'l')
```



Step 3

Current Subtask:

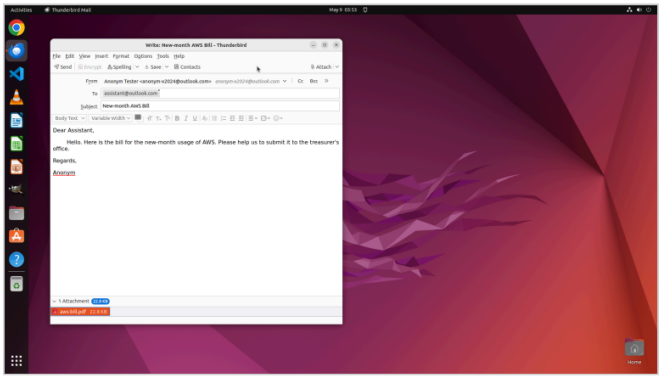
Locate aws-bill.pdf

Action:

click(start_box='(739,175)')

Code:

```
import pyautogui; pyautogui.click(739,
175); pyautogui.write('~/aws-bill.pd
f'); pyautogui.press('ent
```



Step 4 Expert hotkey

Current Subtask:

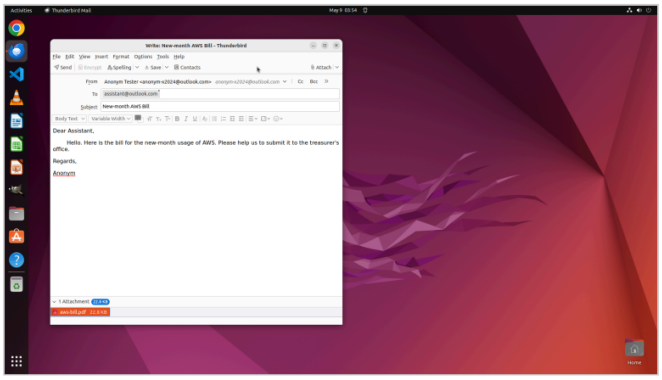
Locate aws-bill.pdf

Action:

hotkey('enter')

Code:

```
import pyautogui; pyautogui.hotkey('ent
er')
```



Step 5

Current Subtask:

Keep compose window open

Action:

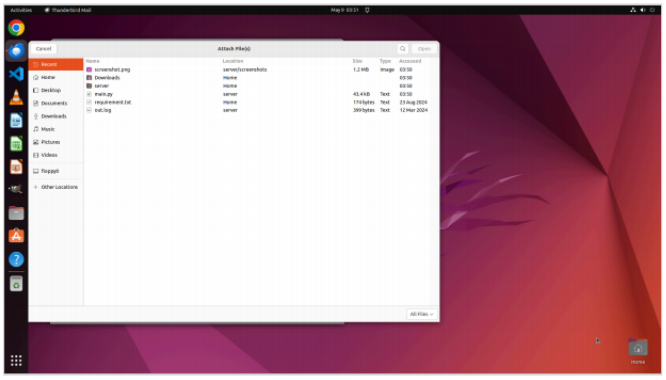
DONE

Code:

```
DONE
```

Converted Trace (from S2)

5 steps → Inherits expert actions



Step 1 Inherited hotkey

Thought:

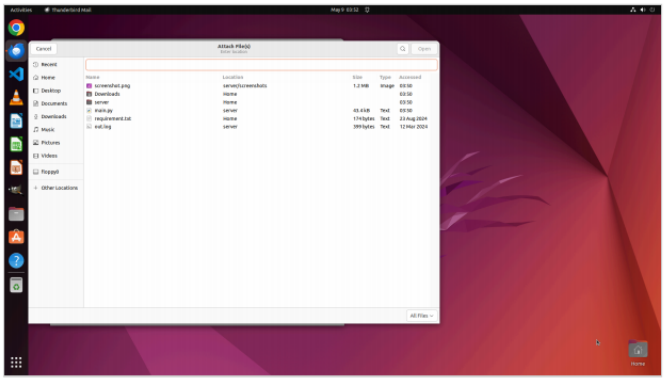
I see that Thunderbird is open with the email compose window active. To attach a file, I need to open the file chooser dialog using the appropriate shortcut. Let me use the Ctrl+Shift+A hotkey to do this.

Action:

hotkey(key='ctrl shift a')

Code:

```
pyautogui.hotkey('ctrl', 'shift', 'a')
```



Step 2 Inherited hotkey

Thought:

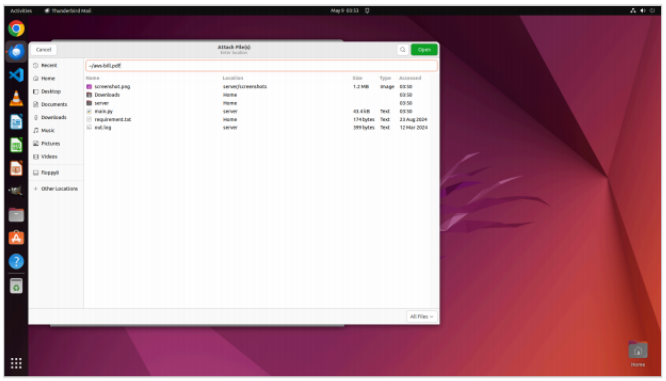
I notice the "Attach File(s)" dialog is open in Thunderbird, and I need to focus on the location bar to input a file path directly. Let me press the Ctrl + L hotkey to focus the location bar.

Action:

hotkey(key='ctrl l')

Code:

```
pyautogui.hotkey('ctrl', 'l')
```



Step 3

Thought:

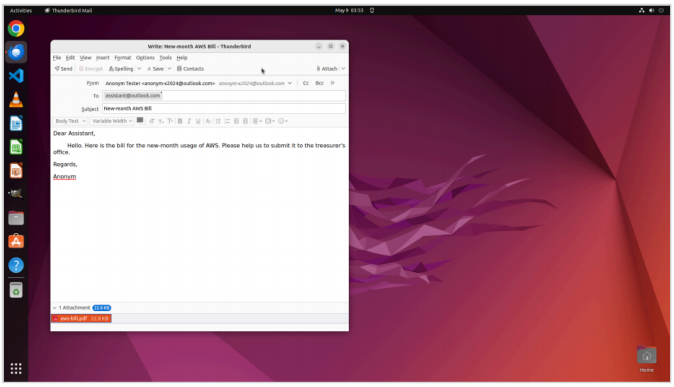
I see the "Attach File(s)" dialog is open and the location bar is focused, indicating it's ready to receive input. To continue, I need to type the file path directly into the location bar and hit Enter to navigate to the specified file.

Action:

click(start_box='<|box_start|>(739,175)<|box_end|>') type(content='~/aws-bill.pdf\n')

Code:

```
pyautogui.click(739, 175)
```



Step 4 Inherited hotkey

Thought:

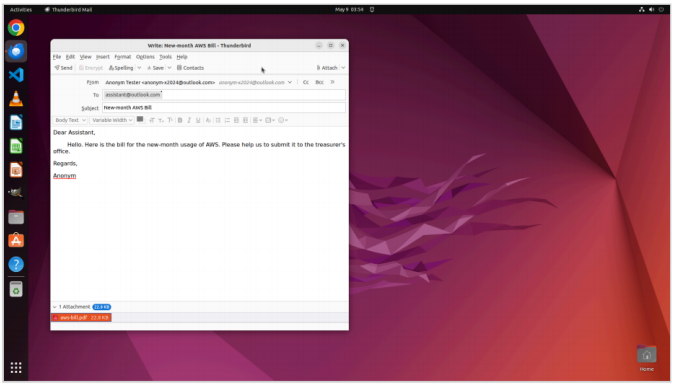
I notice that the file path "~/aws-bill.pdf" is correctly entered in the location bar, but I need to confirm the selection. Let me press Enter to ensure the file picker navigates to the specified file and highlights it.

Action:

hotkey(key='enter')

Code:

```
pyautogui.hotkey('enter')
```



Step 5

Thought:

The task has been completed successfully as per the plan. There are no further actions required, and I can now conclude the operation.

Action:

finished(content='Task completed successfully.\n')

Code:

```
import pyautogui import time ''' Observ
ation: Thought: The task has been compl
eted successfully as
```

Self-Rolled (BEPA L1)

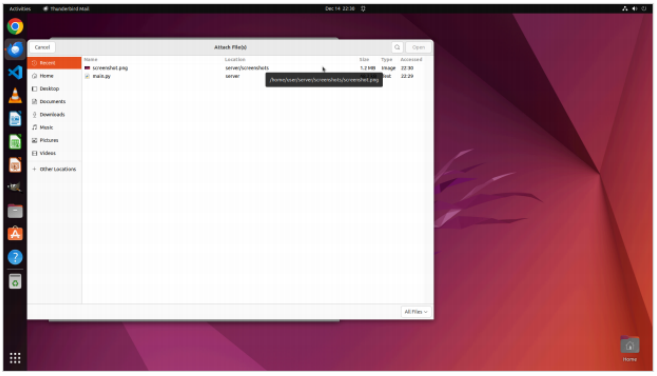
5 steps → Success

Expert Plan (Input)

Task: Attach the AWS bill PDF file to an email in Thunderbird Mail

1. Click on the 'Attach' button in the email composition window
2. Navigate to the 'Home' folder
3. Select the 'aws-bill.pdf' file
4. Click the 'Open' button to attach the file

*Unlike converted traces that inherit expert actions directly, self-rolled traces receive only the expert's **plan** and generate actions through on-policy execution.*



Step 1 On-policy

Thought:

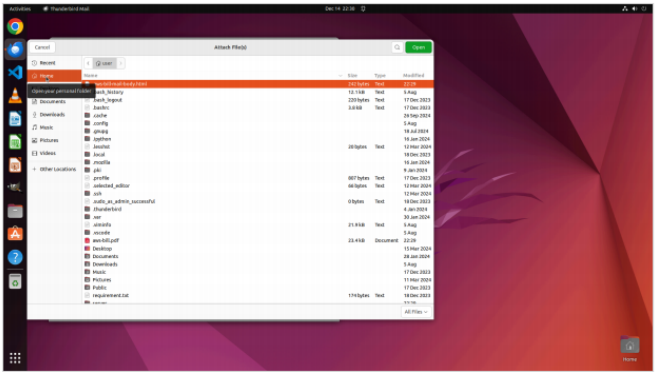
Let me first locate the "Attach" button on the interface. It's positioned towards the upper right side of the window, and it has a downward arrow next to it. I need to click on this button in order to open the file selection window so that I can attach the AWS bill.

Action:

click(start_box='(946,182)')

Code:

```
pyautogui.click(940.124, 180.0, button='left')
```



Step 2

Thought:

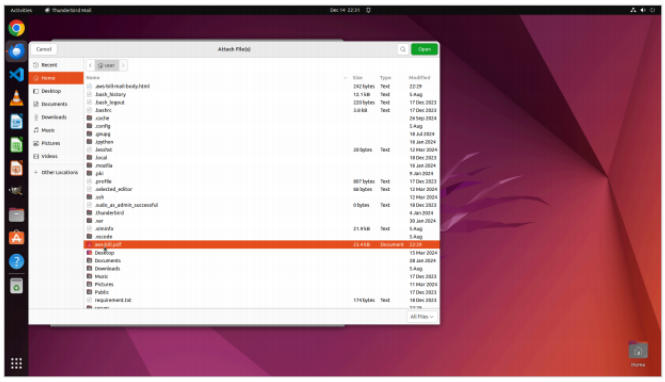
Let's take a look at the current situation. The file selection window is open, and I need to locate the aws-bill.pdf file. Based on my experience, this type of personal file is usually stored in the Home directory. I notice that there's a "Home" option in the left navigation pane, which should be where I need to click next.

Action:

click(start_box='(123,212)')

Code:

```
pyautogui.click(122.236, 209.67, button='left')
```



Step 3

Thought:

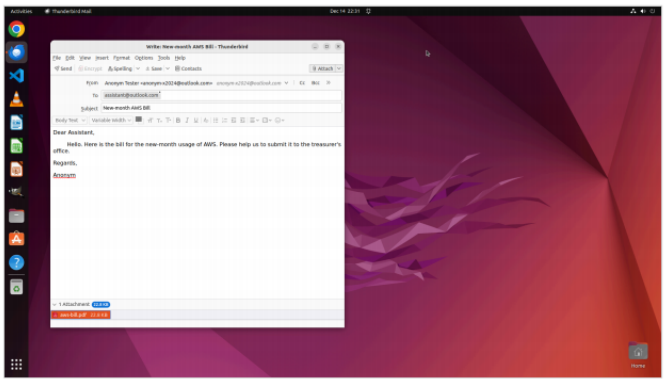
Great, I found the file! The aws-bill.pdf is right in the Home directory. Let's select it and click on "Open" to attach it to the email. This is a crucial step in completing the task.

Action:

click(start_box='(289,705)')

Code:

```
pyautogui.click(287.205, 697.253, button='left')
```



Step 4

Thought:

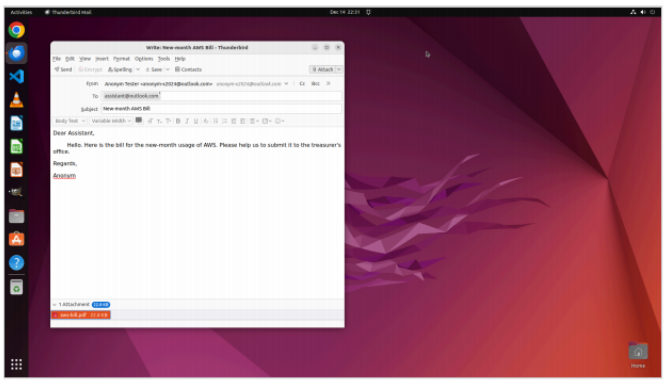
That's fantastic! I've found the aws-bill.pdf file, and it's right in front of me. The file size is 23.4KB, which is exactly what we need to attach. Let's go ahead and click the "Open" button in the upper right corner to add it to the email.

Action:

click(start_box='(1238,126)')

Code:

```
pyautogui.click(1230.311, 124.615, button='left')
```



Step 5

Thought:

That's fantastic! The AWS bill has been successfully attached to the email, and it's showing up in the bottom left corner of the interface. The file size is 22.8KB, which is exactly what we need. It looks like this task has been completed very well, and now we can move on to other work.

Action:

finished(content='已成功将AWS账单附件添加到邮件中，文件大小为22.8 KB。')

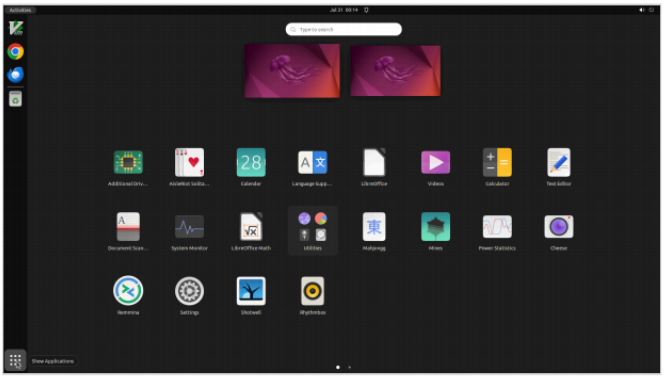
Code:

```
DONE
```

Figure 16: Case study on attaching a file in Thunderbird, comparing Agent S2, a converted trace, and a self-rolled trace (BEPA L1). All three succeed in five steps, but the converted trace inherits S2's hotkeys and template-like chain-of-thought, including references to an external "plan," yielding actions and thoughts that are off the UITARS1.5-7B distribution. The self-rolled trace instead receives only a short expert plan and executes it via on-policy clicks and its own step-wise thoughts, producing a more learnable action and thought distribution.

UITARS1.5-7B

15 steps → Failed



Step 1 Wrong direction

Thought:

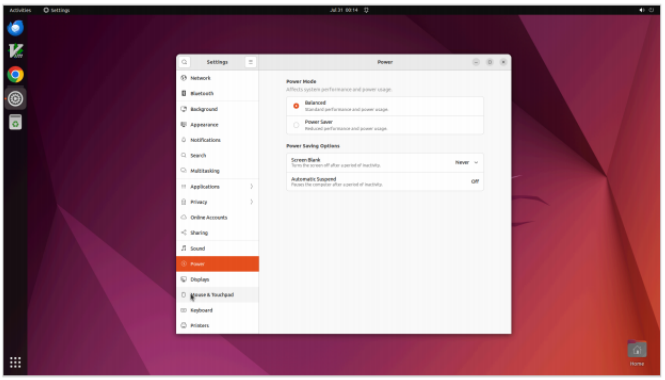
让我看看...要移除vim这个应用，首先得进入系统的应用程序列表。屏幕左下角有个九宫格样式的按钮，点击它就能打开应用程序菜单了。

Action:

click(start_box='(34,1040)')

Code:

```
pyautogui.click(33.789, 1040.44, button='left')
```



Step 2

Thought:

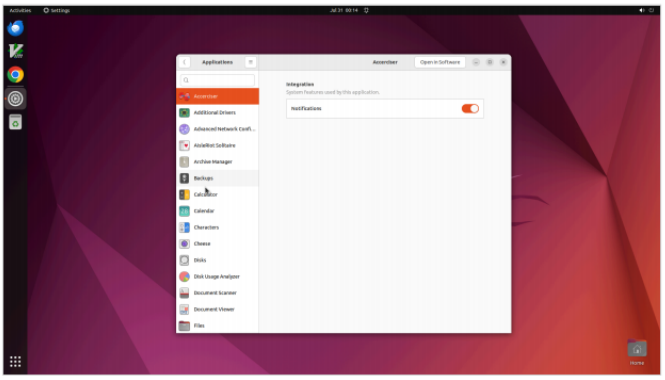
看到这个界面，我需要找到系统设置来管理应用。发现有个齿轮状的"Settings"图标。点击它就能进入系统设置面板了。

Action:

click(start_box='(543,837)')

Code:

```
pyautogui.click(542.609, 836.703, button='left')
```



Step 3

Thought:

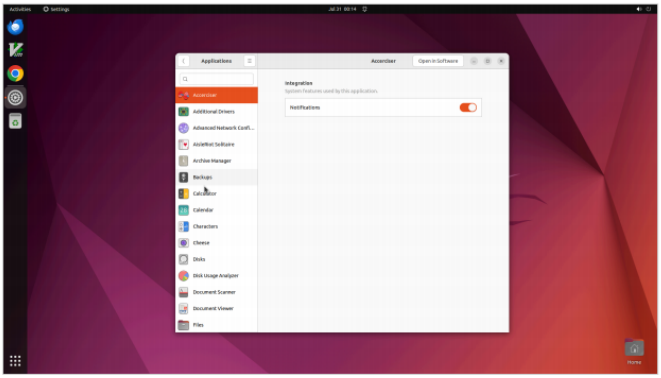
我现在在系统设置界面，要找到管理应用的地方。发现有个"Applications"选项 - 点击它应该就能进到应用管理的页面了。

Action:

click(start_box='(585,526)')

Code:

```
pyautogui.click(585.342, 526.154, button='left')
```



Step 4

Thought:

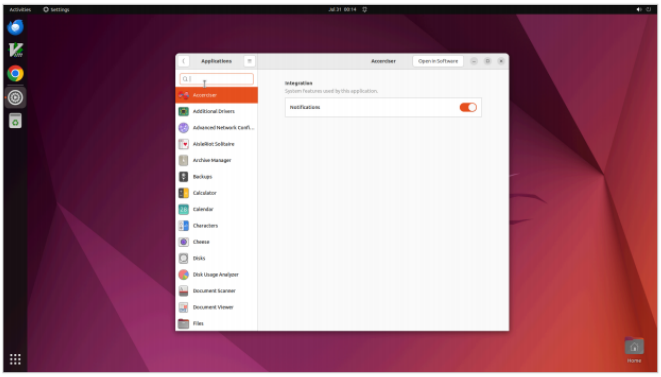
我正在浏览应用程序列表，但目前显示的这些应用里还没看到vim。让我继续往下翻翻看，应该在下面的区域。

Action:

hotkey('pagedown')

Code:

```
pyautogui.hotkey('pagedown')
```



Step 5

Thought:

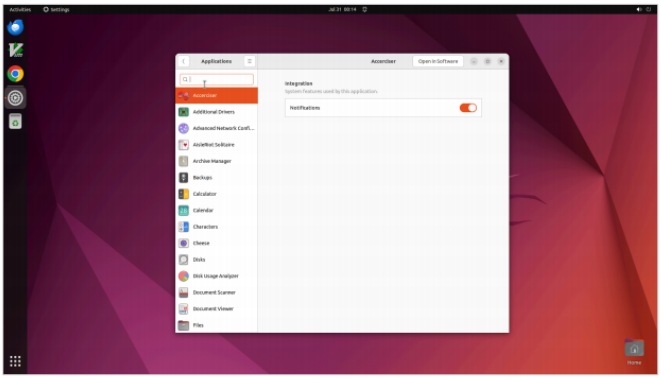
PageDown键似乎不起作用。不过没关系，既然要找vim这个应用，我看到左边有个搜索框，直接在那里输入"vim"应该能更快找到目标。

Action:

click(start_box='(577,218)')

Code:

```
pyautogui.click(577.391, 217.582, button='left')
```



Step 6

Thought:

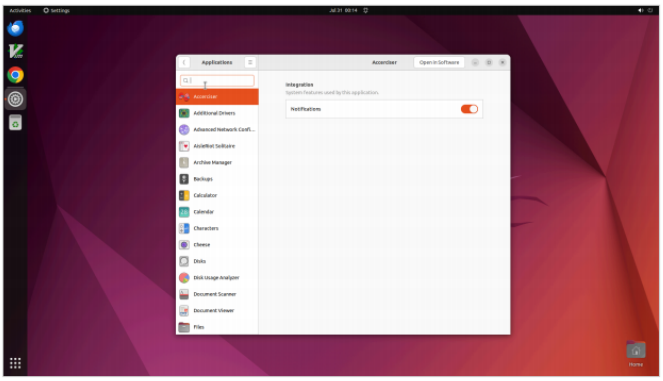
搜索框已经准备就绪了，我看到光标在闪烁。现在只需要输入"vim"这几个字母，系统就会帮我快速定位到这个应用。

Action:

type(content='vim') via Ctrl+V

Code:

```
pyperclip.copy('vim')
pyautogui.hotkey('ctrl', 'v')
```



Step 7

Thought:

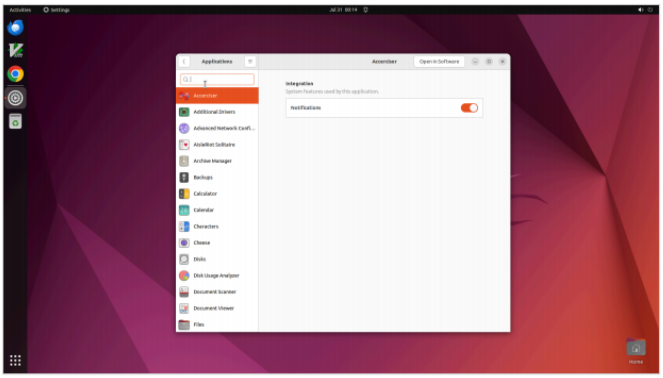
奇怪，我刚才明明在搜索框里输入了"vim"，但怎么没反应呢？看来是系统没有接收到我的输入。没关系，让我再试一次。

Action:

type(content='vim') via Ctrl+V

Code:

```
pyperclip.copy('vim')
pyautogui.hotkey('ctrl', 'v')
```



Step 8

Thought:

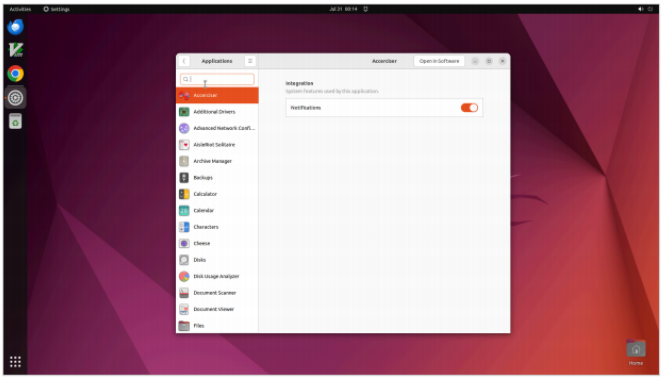
搜索结果已经出来了，我看到Vim的图标就在眼前。要继续完成移除操作，我得先点击这个应用进入它的详细设置页面。

Action:

click(start_box='(541,264)')

Code:

```
pyautogui.click(540.621, 264.066, button='left')
```



Step 9

Thought:

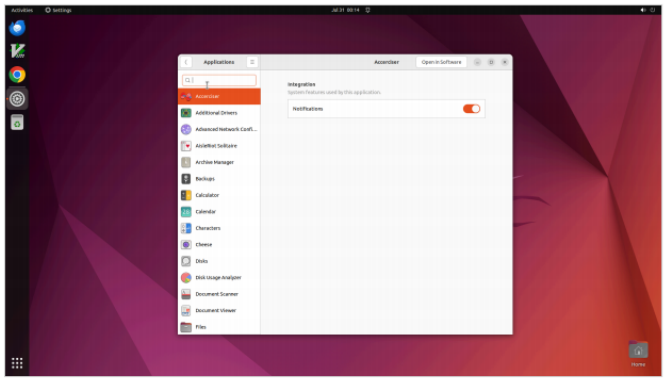
我仔细检查了Vim的设置界面，发现这里只显示了一些基本配置项。看来系统并没有提供直接移除应用的选项。既然这条路走不通，我得换个思路。

Action:

click(start_box='(528,165)')

Code:

```
pyautogui.click(527.702, 165.165, button='left')
```



Step 10

Thought:

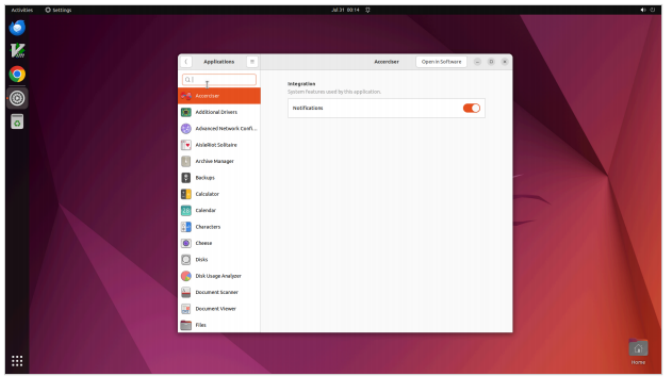
刚才我点击了左上角的返回按钮，但这个操作似乎没有达到预期效果。让我重新思考一下：要移除Vim，我需要回到应用程序列表页面。

Action:

click(start_box='(528,165)')

Code:

```
pyautogui.click(527.702, 165.165, button='left')
```



Step 11

Thought:

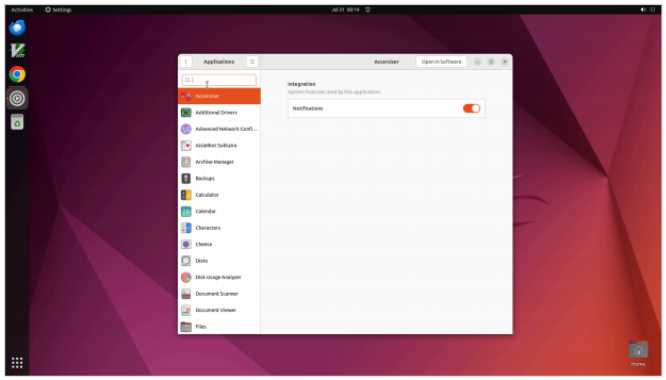
我刚才在设置界面里瞎转悠，完全走错路了。现在我看到右上角有个"Open in Software"按钮，这应该才是正确的方向 - 通过软件中心来管理应用。

Action:

click(start_box='(1276,163)')

Code:

```
pyautogui.click(1276.025, 163.187, button='left')
```



Step 12

Thought:

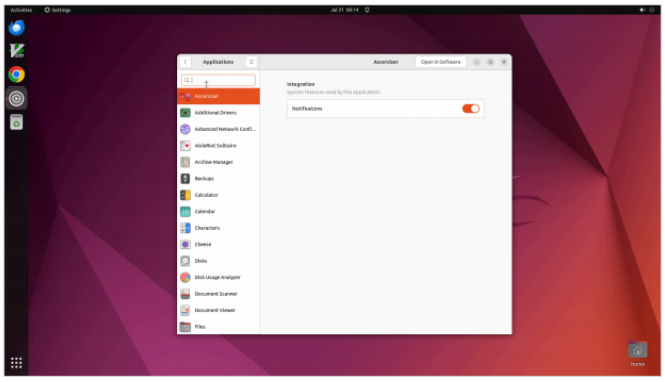
我现在在软件中心界面，看到顶部有个搜索框。要找到Vim并移除它，第一步当然是要先定位到这个应用。

Action:

click(start_box='(91,45)')

Code:

```
pyautogui.click(91.429, 44.505, button='left')
```



Step 13

Thought:

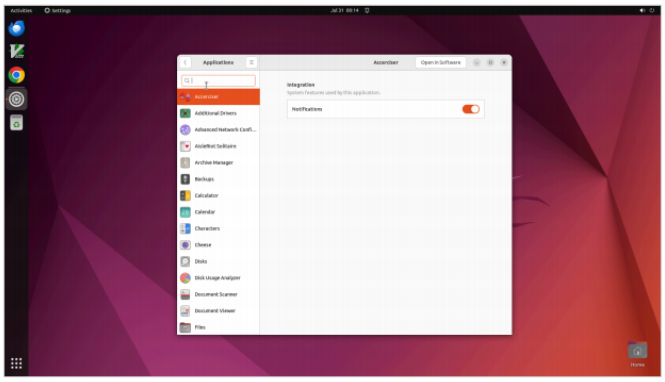
好的，搜索框已经准备就绪了。我需要在这里输入"vim"来找到这个应用。虽然之前在设置界面里没找到移除选项，但**软件中心应该能帮我完成这个任务**。

Action:

type(content='vim') via Ctrl+V

Code:

```
pyperclip.copy('vim')
pyautogui.hotkey('ctrl', 'v')
```



Step 14

Thought:

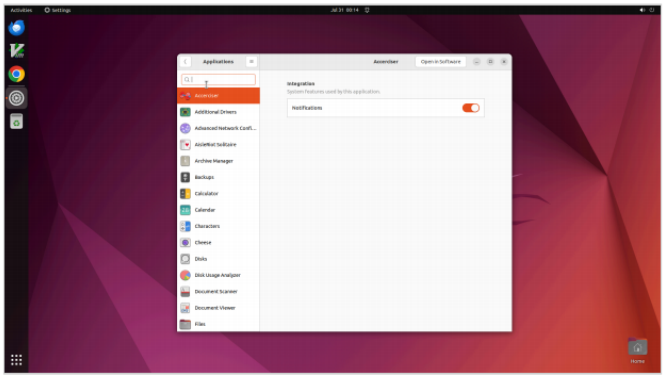
嗯...看来搜索框有点不太配合啊。我刚才输入"vim"，但页面完全没反应。不过没关系，这种情况时有发生，多半是系统响应延迟了。让我再试一次。

Action:

type(content='vim') via Ctrl+V

Code:

```
pyperclip.copy('vim')
pyautogui.hotkey('ctrl', 'v')
```



Step 15 Task Failed

Thought:

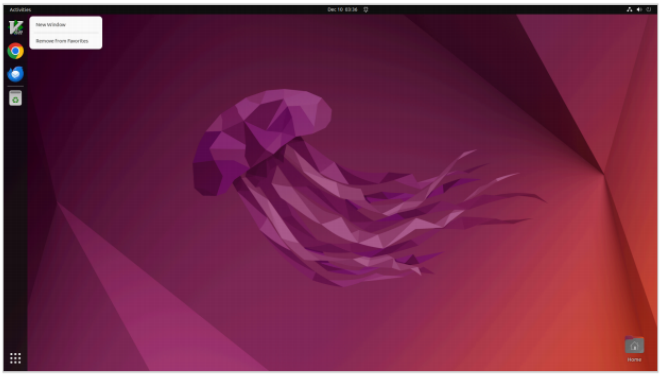
FAIL - 任务失败。从未理解"favorites"指的是dock/收藏栏，一直试图通过系统设置和软件中心来卸载应用。

Action:

FAIL

Agent S2 (Expert Framework)

3 steps → Success



Step 1 Correct understanding

Goal Plan:

1. **Right-click on the Vim icon in the favorites bar (dock)** on the left side of the screen.
2. Click on "Remove from Favorites" in the context menu.

Current Subtask:

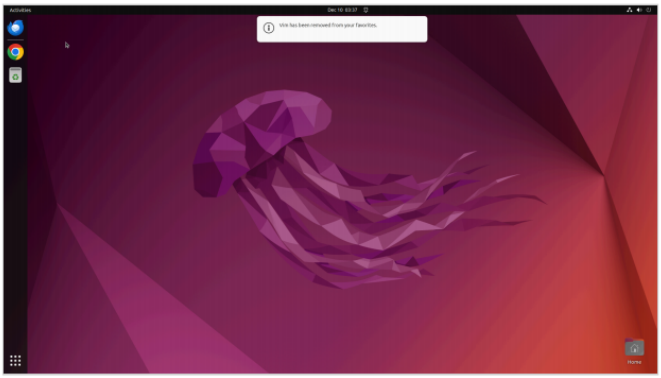
Right-click Vim icon in favorites bar

Action:

click(start_box='**(36,65)**', button='right')

Code:

```
pyautogui.click(36, 65, button='right')
```



Step 2

Current Subtask:

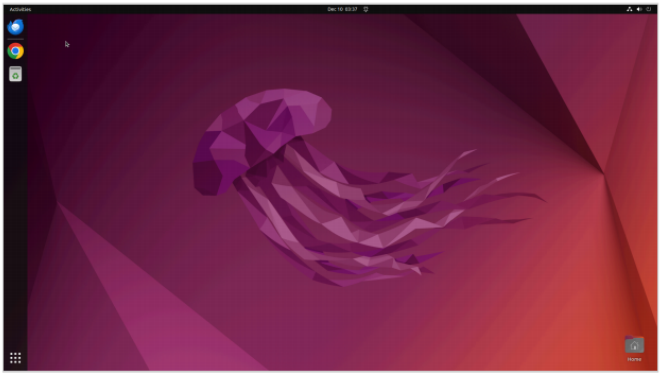
Select 'Remove from Favorites' from context menu

Action:

click(start_box='**(179,104)**')

Code:

```
pyautogui.click(179, 104, button='lef
t')
```



Step 3 Task Completed

Observation:

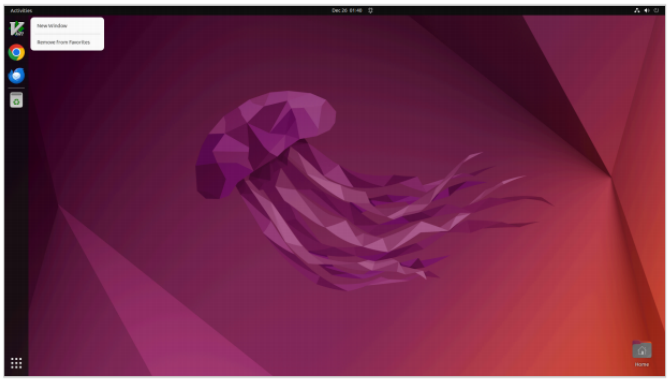
Task completed - Vim has been removed from the favorites bar.

Action:

DONE

BEPA (Ours)

3 steps → Success



Step 1 Learned from expert

Thought:

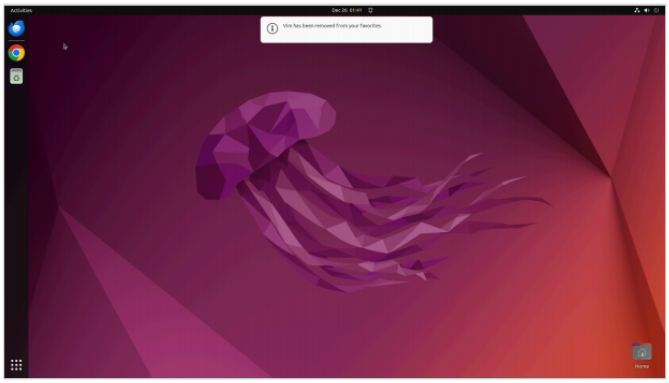
我注意到左侧栏上有个vim的图标，要把它**从收藏夹中移除**的话，我得先**右键点击这个图标**。这样就能打开上下文菜单，看看有什么删除相关的选项。

Action:

click(start_box='**(34,64)**', button='right')

Code:

```
pyautogui.click(33.789, 64.286, button
='right')
```



Step 2

Thought:

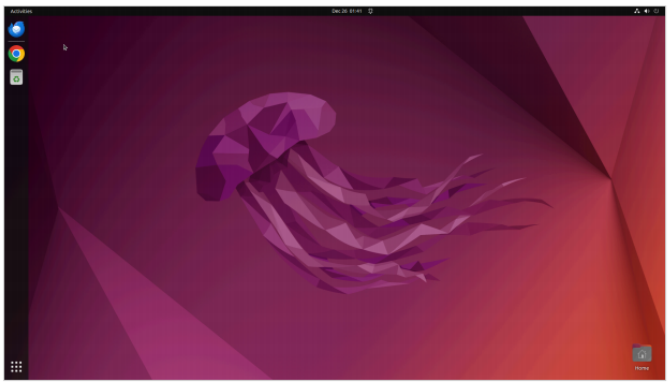
太好了，右键菜单已经弹出来了。我看到了**"Remove from Favorites"**这个选项，这正是我要找的。现在只需要点击这个选项，就能把vim从收藏夹中移除了。

Action:

click(start_box='**(170,107)**')

Code:

```
pyautogui.click(169.938, 106.813, butto
n='left')
```



Step 3 Task Completed

Thought:

Task completed - Vim has been removed from the favorites list.

Action:

DONE

Figure 17: Case study on the OS task "Remove Vim from the favorites list." UITARS1.5-7B misinterprets "favorites" and explores irrelevant settings, while Agent S2 and BEPA both ground it as the dock and quickly remove Vim from the favorites bar. Note that although we emphasized in the system prompt that the agent should output English, the base agent UITARS1.5-7B may still occasionally generate Chinese content due to the characteristics of UITARS's training data, etc.