ByteDance | Seed

UI-TARS-2 Technical Report: Advancing GUI Agent with Multi-Turn Reinforcement Learning

ByteDance Seed

See Contributions section for a full author list.

Abstract

The development of autonomous agents for graphical user interfaces (GUIs) presents major challenges in artificial intelligence. While recent advances in native agent models have shown promise by unifying perception, reasoning, action, and memory through end-to-end learning, open problems remain in data scalability, multi-turn reinforcement learning (RL), the limitations of GUI-only operation, and environment stability. In this technical report, we present UI-TARS-2, a native GUI-centered agent model that addresses these challenges through a systematic training methodology: a data flywheel for scalable data generation, a stabilized multi-turn RL framework, a hybrid GUI environment that integrates file systems and terminals, and a unified sandbox platform for large-scale rollouts. Empirical evaluation demonstrates that UI-TARS-2 achieves significant improvements over its predecessor UI-TARS-1.5. On GUI benchmarks, it reaches 88.2 on Online-Mind2Web, 47.5 on OSWorld, 50.6 on WindowsAgentArena, and 73.3 on AndroidWorld, outperforming strong baselines such as Claude and OpenAI agents. In game environments, it attains a mean normalized score of 59.8 across a 15-game suite-roughly 60% of human-level performance-and remains competitive with frontier proprietary models (e.g., OpenAI o3) on LMGame-Bench. Additionally, the model can generalize to long-horizon information-seeking tasks and software engineering benchmarks, highlighting its robustness across diverse agent tasks. Detailed analyses of training dynamics further provide insights into achieving stability and efficiency in large-scale agent RL. These results underscore UI-TARS-2's potential to advance the state of GUI agents and exhibit strong generalization to real-world interactive scenarios.

Correspondence: yujia.qin@bytedance.com; shiguang.sg@bytedance.com

[a]Project: https://github.com/bytedance/ui-tars, https://github.com/bytedance/UI-TARS-desktop
[b]Demo: https://seed-tars.com/showcase/ui-tars-2

Contents

1 Introduction

The development of agents that can operate seamlessly within graphical user interfaces (GUIs) has emerged as a central challenge in artificial intelligence [25, 40, 63, 70, 87]. Traditional approaches typically adopt modular pipelines with separately engineered components for perception, planning, memory, and action [34]. While such design-driven systems enable rapid progress in specific domains, they rely heavily on expert heuristics and task-specific rules, leaving them brittle and difficult to scale. Recent work on *native agent models* [49] shifts toward data-driven, end-to-end learning, where perception, reasoning, and control are unified within a single policy, offering a more scalable and adaptive path for GUI agents.

Despite recent progress, the development of robust GUI agents still faces several open challenges. **(1) Data scarcity.** While large-scale pre-training and reinforcement learning have proven effective in reasoning and chat domains [27, 41], scalable strategies for long-horizon GUI learning remain unclear. Unlike text or code corpora, large-scale trajectories that capture detailed reasoning, actions, environment states, and feedback are extremely costly to collect. **(2) Scalable multi-turn RL.** RL in interactive environments is notoriously difficult: rewards are often sparse or delayed, optimization can be unstable, and credit assignment across long sequences of actions remains challenging. These issues hinder scaling beyond short-horizon demonstrations and make it hard to achieve stable improvements in complex tasks. **(3) Limitations of GUI-only operation.** Pure GUI interaction is often insufficient for realistic workflows. Many tasks—such as data processing, software development, or system administration—are more naturally handled through file systems, terminals, or external tools rather than by simulating mouse clicks and keystrokes. Thus, advancing GUI agents requires environments that allow graphical actions to interoperate seamlessly with other resources, broadening the scope of tasks they can effectively solve. **(4) Environment scalability and stability.** Even with richer interaction capabilities, deploying large-scale RL environments remains an engineering bottleneck. Rollouts must be reproducible, fault-tolerant, and capable of supporting millions of interactive episodes across browsers, VMs, and simulators. In practice, such environments are fragile, resource-intensive, and prone to crashes, making stable large-scale training particularly challenging.

To address these challenges, we introduce a systematic methodology built on four pillars. First, to mitigate data scarcity, we design a scalable **Data Flywheel** that co-evolves the model and its training corpus through continual pre-training, supervised fine-tuning, rejection sampling, and multi-turn RL. This framework supplies a steady stream of diverse, high-quality trajectories and ensures that both the model and the data improve iteratively in a self-reinforcing cycle. Second, to overcome the difficulties of **scalable multi-turn RL**, we design a training framework that stabilizes optimization in long-horizon settings. This includes asynchronous rollouts with stateful environments to preserve context, streaming updates to avoid bottlenecks from long-tail trajectories, and enhanced proximal policy optimization [55] with reward shaping, adaptive advantage estimation, and value pretraining. Third, to move beyond the limitations of pure GUI interaction, we construct a **hybrid GUI-centered environment** that augments on-screen actions with access to complementary resources such as file systems, terminals, and other external tools, enabling agents to solve a broader spectrum of realistic workflows. Fourth, to support large-scale training and evaluation, we build a **unified sandbox platform** capable of orchestrating heterogeneous environments—ranging from cloud VMs for GUI interaction to browser-based sandboxes for games—under a consistent API. The platform is engineered for reproducibility, stability, and high throughput, making it possible to run millions of interactive rollouts reliably.

Empirical evaluation shows that UI-TARS-2 delivers significant improvements over UI-TARS-1.5 [56], achieving strong results in both GUI-based interaction and game environments. On GUI benchmarks, the model reaches 88.2 on Online-Mind2Web [77], 47.5 on OSWorld [75], 50.6 on WindowsAgentArena [10], and 73.3 on AndroidWorld [52], representing clear gains over the previous generation and outperforming strong baselines such as Claude and OpenAI agents in multiple cases. In game environments, UI-TARS-2 attains a mean normalized score of 59.8 across a 15-game suite—roughly 60% of human-level performance—and surpasses strong baselines such as OpenAI CUA and Claude Computer Use by factors of 2.4× and 2.8×, respectively. On LMGame-Bench [24], UI-TARS-2 remains competitive with frontier proprietary models, further highlighting its robustness in long-horizon game reasoning. Beyond GUI and games, we extend the agent's capabilities through GUI-SDK, enabling integration with system-level resources such as terminals and external tools. With this extension, UI-TARS-2 demonstrates strong performance on long-horizon information-seeking benchmarks (e.g.,

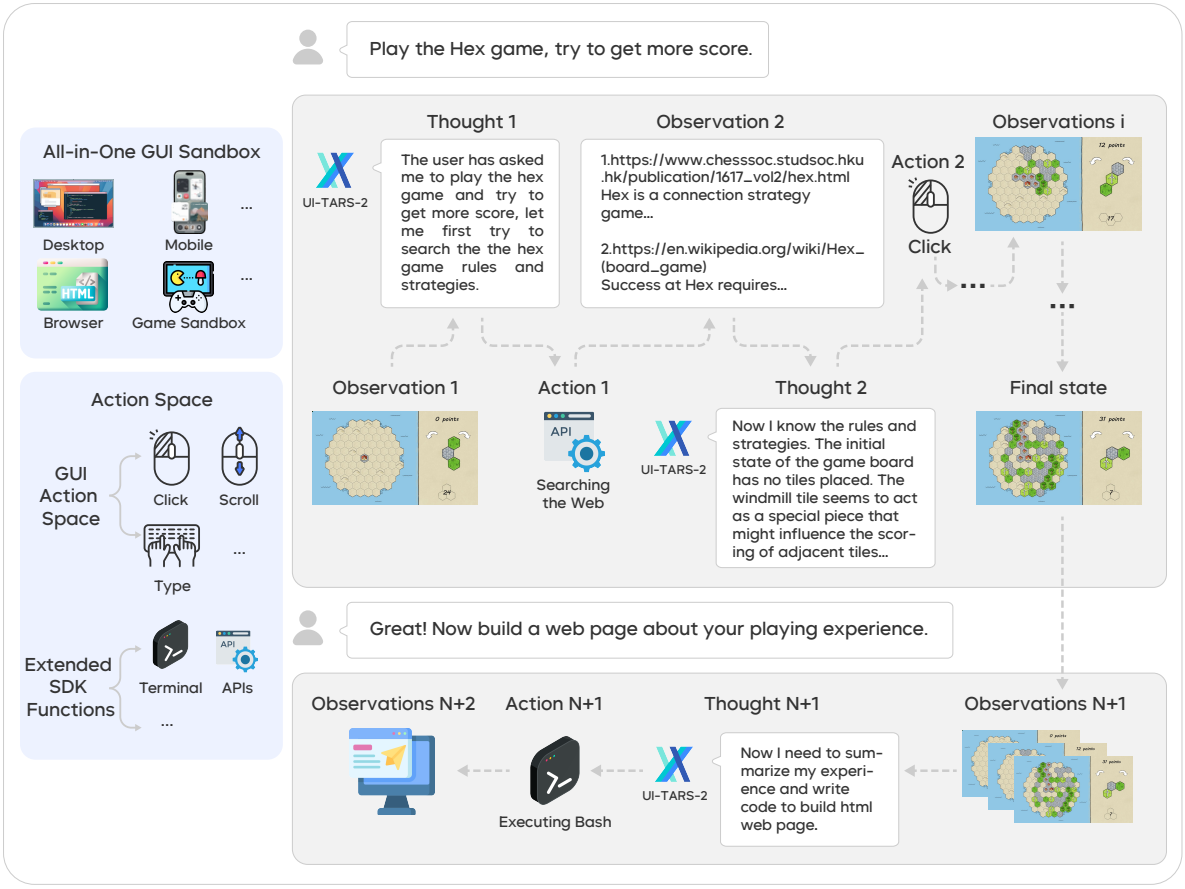


Figure 1 A demo trajectory of UI-TARS-2.

29.6 on BrowseComp [73]) and competitive results on software engineering tasks (45.3 on Terminal Bench [66], 68.7 on SWE-Bench Verified [28]). These results suggest that the training methodology developed for GUI agents—particularly multi-turn RL optimization and scalable rollout infrastructure—transfers effectively to other interactive domains, broadening the agent's applicability. In addition, our detailed analyses of training dynamics, interaction scaling, and hybrid training strategies provide practical insights into achieving stability and efficiency in large-scale agent reinforcement learning. Taken together, these findings establish UI-TARS-2 as a robust GUI-centered agent that not only advances the state of GUI interaction but also generalizes effectively to diverse real-world environments.

2 UI-TARS-2

This section introduces the methodology of UI-TARS-2, a unified framework for building advanced GUI-centered agents. We illustrate a demo trajectory of UI-TARS-2 in Figure 1. Our approach integrates multiple components, including formal agent formulation, all-in-one sandbox environments, a data flywheel pipeline, multi-turn reinforcement learning, and parameter interpolation across vertical agents.

2.1 Formulation

We adopt a native agent perspective [49], where an agent is modeled as a parameterized policy that maps historical context, memory states, and the current environment into behavioral outputs. At timestep t, the agent follows the ReAct paradigm [79], which interleaves reasoning, action, and observation in a structured loop:

- **Reasoning (t_t)**: internal cognitive processing, including context analysis, memory recall, planning, and self-reflection.
- **Action (a_t)**: external interaction, such as GUI manipulation, system commands, or tool invocation.
- **Observation (o_t)**: feedback from the environment used to update the agent's state.

Our action space spans multiple categories of operations:

- **GUI Actions**: direct interface manipulation following UI-TARS [49], e.g., clicks for element selection, typing for text input, and scrolling for navigation. Gameplay interactions also reuse these same primitives.
- **Pre-defined SDK Functions**: supplementary operations that extend beyond GUI manipulation, including direct terminal commands for file management and software development, as well as MCP tool invocations for orchestrating external services and multi-tool reasoning.

We define a *step* as one complete ReAct cycle (t_t, a_t, o_t). A trajectory of length T is then formulated as:

$$\tau = \{(t_0, a_0, o_0), (t_1, a_1, o_1), \ldots, (t_T, a_T, o_T)\}. \tag{1}$$

A key component of this formulation is the hierarchical memory state:

$$\mathcal{M}_t = (\mathcal{W}_t, \mathcal{E}_t), \tag{2}$$

where **Working Memory** $\mathcal{W}_t$ stores recent steps $(t_{t-k}, a_{t-k}, o_{t-k})$ in high fidelity for short-term reasoning, while **Episodic Memory** $\mathcal{E}_t$ maintains semantically compressed summaries of past episodes, preserving key intentions, and outcomes. To remain efficient under long trajectories, we restrict direct context to the last N steps from $\mathcal{W}_t$, while conditioning on $\mathcal{E}_t$ for longer-term recall. At each timestep, the policy predicts the next thought and action as:

$$P(t_n, a_n \mid \text{instruction}, \mathcal{W}_n, o_n, \mathcal{E}_n). \tag{3}$$

This highlights that agent behavior arises not from isolated predictions, but from an evolving loop of reasoning, action, feedback, and memory integration.

2.2 Environment: All-in-One GUI Sandbox

Training a general-purpose computer agent that seamlessly integrates a a wide range of computational capabilities imposes exceptionally demanding environmental requirements. Unlike single-domain simulators, such new environments must support diverse task types, integrate heterogeneous tools, and preserve long-lived state across complex, multi-step interactions.

To address these challenges, we engineered a universal sandbox that merges GUI operations and SDK functions (e.g., file system and tool calling) into a cohesive and versatile platform. A core innovation is the shared file system, which allows an GUI agent to, for instance, download a file via the browser and immediately process it using shell commands within the same containerized instance. The sandbox maintains the stability and reproducibility essential for complex tasks and allows for not only high-throughput training on a distributed computing backbone, but also a consistent environment for annotation, evaluation, and inference. Here we highlight the design of GUI and game sandbox.

GUI Env: Cloud Virtual Machine. To support large-scale training and evaluation of GUI agents, we developed a distributed virtual machine (VM) platform that runs mainstream desktop operating systems (Windows and Ubuntu) as well as the Android mobile OS. The platform integrates PyAutoGUI and ADB interfaces, enabling cross-device operations with minimal adaptation overhead. A unified SDK standardizes the entire interaction pipeline—from VM allocation and initialization to agent interaction, observation collection (e.g., screenshots and recordings), and task evaluation—making the system suitable for diverse use cases such as manual data annotation, OSWorld benchmarking, and online reinforcement learning.

At the infrastructure level, the VM cluster comprises several thousand instances, centrally managed by a VM Manager capable of sustaining throughput at several thousand QPS (Queries Per Second) and handling

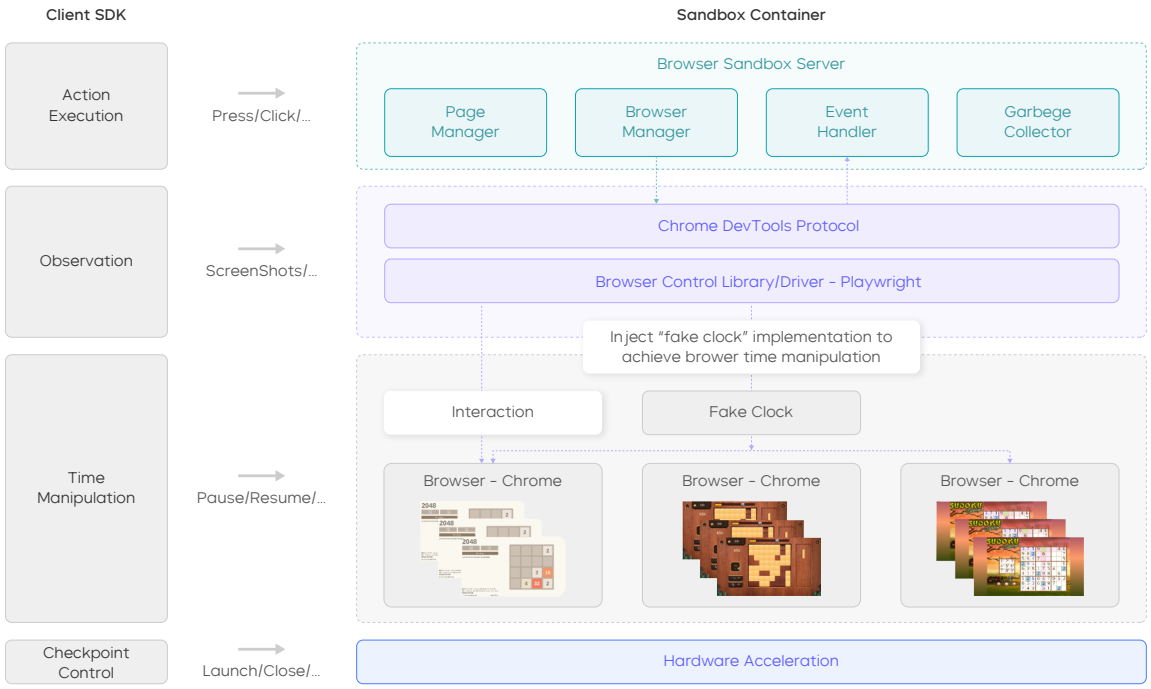


Figure 2 Browser sandbox (container) architecture.

high-concurrency execution. Each session is tracked with a task–environment mapping via session IDs to ensure state consistency across multi-round interactions. For monitoring and control, all sessions are visualizable in real time via VNC (Virtual Network Computing) / RTC (Real-Time Communication). A lease-based lifecycle mechanism automatically releases resources after task completion or failure, while overdue sessions are reclaimed to prevent waste.

Beyond GUI interaction, the platform extends the agent's capabilities with tool calling and coding support, enabling cross-domain workflows such as web browsing, file manipulation, and software development. An integrated endpoint pre-loads essential local services for browsing, file access, and terminal use, ensuring that tools are available out-of-the-box. The sandbox also enhances the coding environment by allowing services launched from the terminal to be exposed via proxy URLs, enabling the GUI agent to preview both front-end and back-end components. For human-in-the-loop debugging and annotation, the environment further provides VNC, a remote VS Code editor, Jupyter, and terminal previews directly in the browser.

Game Env: Hardware-Accelerated Browser Sandbox. To support high-throughput rollouts for multi-turn RL on web-based mini-games, we built a browser sandbox that serves as the execution and observation backbone. Because these mini-games run entirely in HTML5/WebGL, a browser environment is the only practical way to execute them faithfully while capturing their full interactive state. The sandbox exposes unified "page management + page interaction" APIs: clients issue actions (e.g., keyboard/mouse inputs) and receive synchronous observations (screenshots, scores, levels), completing the standard action-to-state loop.

As illustrated in Figure 2, concurrency is achieved by running multiple browser instances per container with elastic scheduling. The system monitors main processes and performs automatic crash recovery to ensure long-running stability. A page-control layer manages page creation and deletion, maintains session–page mappings, tracks page states, and executes commands, while checkpointing ensures reproducibility. An event handler continuously reports browser/page events to the manager, and a garbage collector reclaims idle sessions to prevent resource leakage.

For programmatic access, the sandbox is compatible with the Chrome DevTools Protocol and popular drivers

such as Playwright, enabling orchestrated, debuggable, and auditable interaction. GPU-based hardware acceleration reduces screenshot overhead, while re-implemented Window timing APIs allow time acceleration and pause at startup, improving sampling efficiency and reproducibility without altering game logic. In sum, the sandbox functions like a standard RL environment but is engineered specifically for the web stack, balancing high concurrency, determinism, and reproducibility.

2.3 Data Flywheel Overview

As shown in Figure 3, we introduce the data flywheel that continually improves both model capabilities and data quality through repeated training cycles. In each cycle, the latest model generates new agent trajectories, which are filtered and redistributed to the most suitable training stages. High-quality outputs are promoted to later stages (e.g., SFT), while lower-quality outputs are recycled into earlier stages (e.g., CT). Over successive iterations, this dynamic reallocation ensures that every stage operates on optimally matched data, creating a self-reinforcing loop where better models yield better data, and better data produces better models.

Training Stages. Starting from the pre-trained checkpoints of Seed1.6 [11], the flywheel operates through three stages: continual pre-training (CT) — broad knowledge acquisition from large-scale, diverse data, supervised fine-tuning (SFT) — high-quality, task-specific instruction tuning, and reinforcement learning — end-to-end optimization on verifiable interactive tasks. In each iteration, the current RL model generates new trajectories. High-quality outputs are appended to the SFT dataset, lower-quality ones are routed to CT, and the model is retrained sequentially on the updated CT, SFT, and RL stages.

Cold-start Data Sources. The flywheel is bootstrapped with two initial datasets. For CT, we collect task tutorials, instructional videos, demonstrations from the internet, and our in-house data (section 2.4.1) to form the base knowledge set $D_{CT}^{(0)}$. For SFT, we construct $D_{SFT}^{(0)}$ through synthetic data generation and human annotation. During both CT and SFT, agent-specific data is mixed with general-purpose data, including chat and reasoning domains. Agent-specific data constitutes only a small fraction of CT, which emphasizes broad knowledge acquisition. In contrast, agent data forms a much larger proportion of SFT, which focuses on high-quality, task-specific agent trajectories.

Iterative Data Flow. After the initial RL model is trained, it becomes the main data generator for the next iteration. In each iteration t, it produces new trajectories via rejection sampling (RFT) or interactive annotation (section 2.4.2). Each sample is evaluated by a validation function $V(s) \to \{0, 1\}$. High-quality samples with $V(s) = 1$ are added to the SFT dataset for the next iteration as $D_{SFT}^{(t+1)} = D_{SFT}^{(t)} \cup D_{\text{RFT,high}}^{(t)}$, while lower-quality samples with $V(s) = 0$ are routed to the CT dataset as $D_{CT}^{(t+1)} = D_{CT}^{(t)} \cup D_{\text{RFT,low}}^{(t)}$. This ensures that SFT always receives the most recent, verified high-quality data, while CT continually expands with broader, less polished knowledge without contaminating the supervised signal. Note SFT and RL are performed more frequently than CT. It should also be noted that in each cycle, we observe substantial transfer from general-purpose RL to agent-specific domains. As iterations progress, the improved model $M^{(t+1)}$ generates a higher proportion of high-quality outputs, i.e., $P(V(s) = 1 \mid t) > P(V(s) = 1 \mid t-1)$, accelerating capability growth. Since every generated sample is reused at an appropriate stage, no data is wasted, creating a sustainable cycle in which model and data quality co-evolve to drive continual performance gains.

2.4 CT & SFT Data Preparation

Agent-related training data represents a significant scarcity in existing human corpora, particularly for multi-turn interactive tasks that require sustained reasoning and tool manipulation. Unlike abundant mathematical or coding data in human corpora, agent interaction trajectories are rare and difficult to obtain at scale. To address this critical bottleneck, we develop a systematic data construction pipeline that operates on both interactive human annotations and automated data synthesis.

2.4.1 In-Situ Annotation for Continual Pre-training

Our continual pre-training framework spans multiple agent domains. Here we illustrate the methodology using the GUI domain as a representative case. As the cold-start GUI CT dataset $D_{CT,GUI}^{(0)}$, we include all training data from UI-TARS [49] and UI-TARS-1.5 [56], consisting of GUI tutorials collected from the

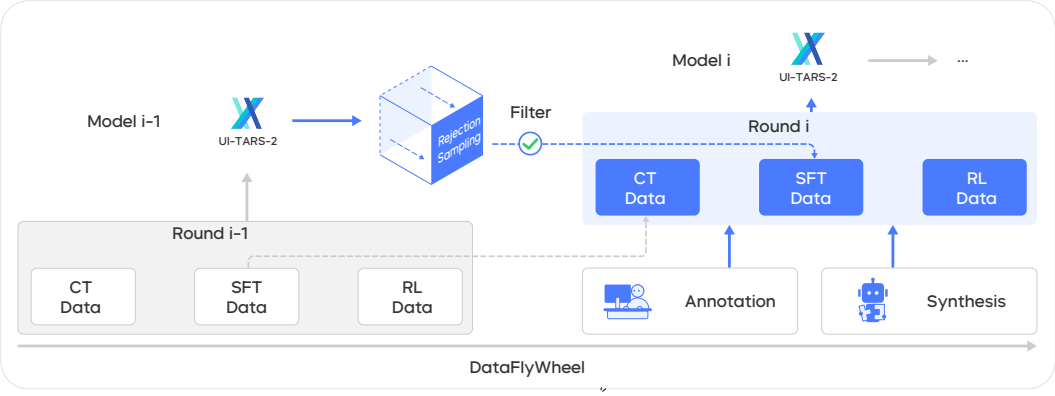


Figure 3 We curate a Data Flywheel for UI-TARS-2, establishing a self-reinforcing loop that continuously improves both data quality and model capabilities.

internet, open-source agent trajectories, our in-house annotations, etc. Despite this diverse initialization, we quickly encountered several limitations. First, publicly available data is inherently scarce and easily exhausted, leaving insufficient coverage for training at scale. In particular, we observed a notable lack of content for Chinese-language applications, which hinders the development of truly versatile agents. Second, much of the available data provides only procedural actions while omitting the underlying cognitive reasoning. Models trained solely on such resources tend to mimic surface-level actions without internalizing the logic, leading to spurious or unstable reasoning chains. Ultimately, the central challenge of continual pre-training lies in how to systematically scale up high-quality, cognitively rich data to sustain long-term agent improvement.

To address the deficiencies of existing GUI datasets, we developed a large-scale, human-centric annotation system designed for collecting authentic cognitive processes. A key feature of our platform is its **in-situ deployment**: the annotation tool is directly installed on annotators' personal computers and runs unobtrusively alongside their normal usage. This design allows data to be collected continuously in realistic, everyday settings, without disrupting natural workflows.

Annotation Protocol. An initial pilot study that attempted to retroactively add reasoning traces to recorded actions proved ineffective, as it was nearly impossible to reconstruct the annotator's original thought process. Inspired by Deitke et al. [15], we instead adopted a *think-aloud* protocol, where annotators verbalize their thoughts via audio while completing tasks. These verbalized thoughts are automatically aligned with corresponding UI interactions, producing data that captures both the reasoning chain and the grounded actions. To further enrich coverage, we recruited two groups of annotators: (1) **experts**, who provide demonstrations of complex tasks, and (2) **novices**, who are asked to solve unfamiliar tasks through exploration, trial-and-error, and external resources (e.g., web search). The novice track captures valuable data on problem-solving and adaptation when prior knowledge is absent.

Task Design and Collection. To strengthen GUI-agent capabilities in realistic settings, we present a reproducible data acquisition pipeline. Candidate applications are selected using publicly available indicators along three dimensions—industry coverage, user engagement, and market penetration—yielding a representative set of mainstream websites and desktop applications. For each service, a hierarchical task graph is constructed, and task-importance scores are derived using normalized measures of usage frequency, user benefit, and cross-scenario transferability. We adopt a human–LLM collaborative workflow to generate multilevel query sets for each subfunction, spanning novice-to-expert skill levels and both single- and multi-application settings. A difficulty rubric based on step count, cross-page operations, prerequisites, and exception handling ensures balanced coverage across difficulty levels.

Curation Pipeline. All collected data undergoes rigorous quality control, including executability verification, deduplication, and dual-annotator review. The audio-recorded thoughts are first transcribed using automatic speech recognition (ASR) and then refined by LLMs to produce coherent, high-quality reasoning text. These processed reasoning traces are precisely synchronized with on-screen actions, yielding temporally aligned

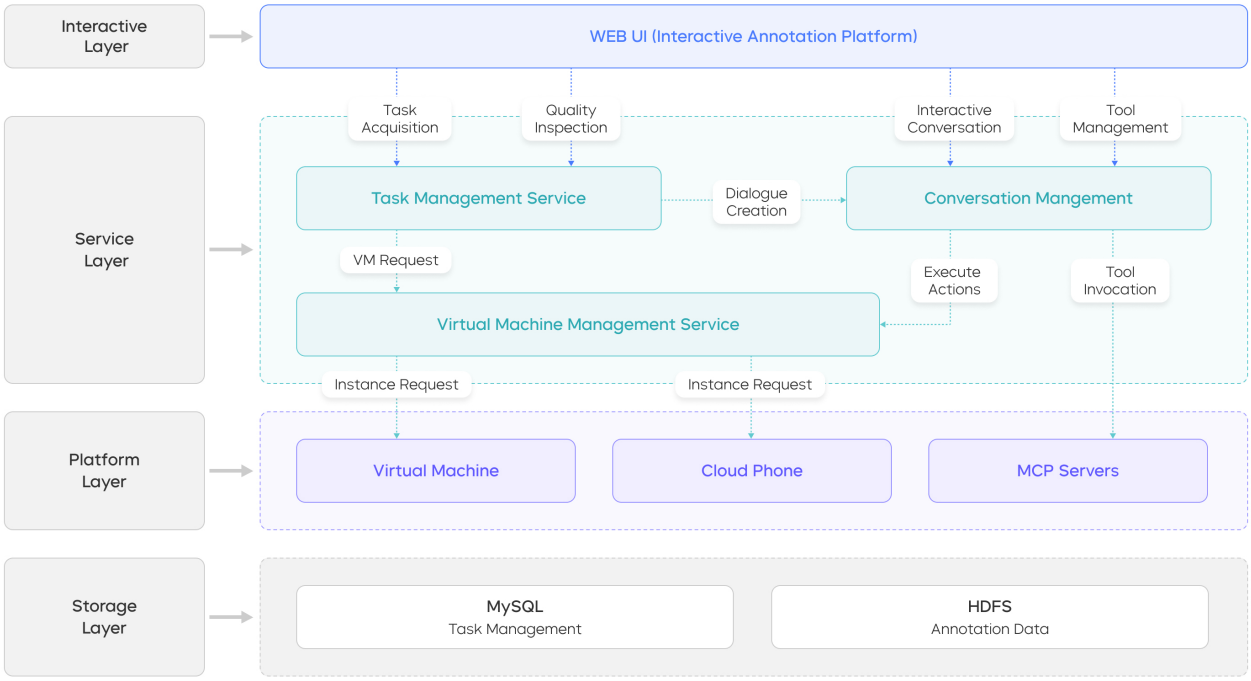


Figure 4 The four-layer architecture of the interactive annotation platform.

reasoning–action trajectories. To further enhance training utility, we programmatically augment linguistic diversity and enrich reasoning chains, resulting in a final high-fidelity dataset suitable for continual pre-training.

2.4.2 Interactive Annotation for Supervised Fine-tuning

A key challenge in training agents from human-generated SFT data is that such data is typically *off-policy*: it does not reflect the actual distribution of actions that the model would take when interacting with an environment. As a result, models trained on this data may fail to generalize, since they never encounter or correct their own mistakes during rollout. Prior approaches mitigate this by asking annotators to correct errors in pre-collected trajectories [49]. However, this procedure remains fundamentally offline and inefficient: it exposes model weaknesses only after task failure, without enabling real-time intervention or correction during interaction. Because agent training occurs within interactive environments, where actions directly affect subsequent states, this lack of on-policy supervision creates a significant gap. To bridge it, we propose a novel human-in-the-loop framework for online, interactive data annotation.

System Design. Our interactive annotation platform is built on a four-layer architecture. At the top, the interaction layer presents the user interface, enabling annotators to engage with the system in real time. Beneath it, the service layer processes annotation requests, orchestrating model-generated command execution and human interventions. The platform layer provides scenario-specific execution environments—such as Computer Use, Phone Use, or Tool Use—tailored to different categories of tasks. Finally, the storage layer securely logs annotation data and complete interaction trajectories for downstream training and analysis. The overall design is illustrated in Figure 4, which depicts the modular separation between layers and their control flow. In the following, we take GUI and Game as examples to illustrate the annotation process.

Our interactive annotation platform enables human annotators to provide online supervision directly within the agent's rollout. Annotators are assigned tasks to complete in a controlled virtual environment (see Figure 5), backed by a cloud-hosted VM or browser sandbox to ensure reproducibility and consistent execution. At each decision point, the latest UI-TARS-2 model proposes candidate actions together with its reasoning trace. The annotator can either accept one of these suggestions or override it with a better thought and action, allowing human expertise to guide the trajectory in real time. We further streamline the workflow

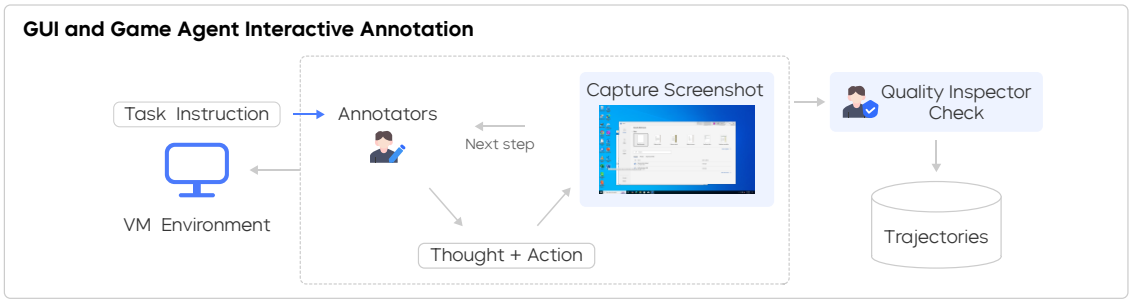


Figure 5 The interactive annotation workflow.

with features such as command auto-completion, real-time VM video streaming, and on-screen coordinate visualization, reducing latency and improving annotation accuracy.

Because annotation occurs in a live environment, annotators receive immediate feedback from the system and can track the evolving trajectory, avoiding the inefficiencies of post-hoc correction. This design ensures that all supervision remains strictly *on-policy*: the data reflects the actual distribution of states visited by the current model. To further enhance efficiency, both the annotation model and the pool of tasks are periodically refreshed, ensuring that data collection consistently targets the weaknesses of the most recent agent.

2.5 Multi-turn Reinforcement Learning

To train agents capable of long-horizon reasoning and interactive decision-making, we adopt a multi-turn RL framework built on RLVR (Reinforcement Learning with Verifiable Rewards) [21]. We construct domain-specific pipelines that automatically synthesize large-scale, verifiable tasks across multiple domains. During RL, our model engages in real-time multi-turn interactions with the environment, continuously observing state transitions and environmental feedback until task completion. The model then leverages verifiable rewards to optimize its decision-making trajectories through iterative policy improvement. While our RL framework is applied across multiple domains with different tools defined by GUI operations and GUI-SDK functions, in the following, **we choose three representative cases to describe our framework**: (1) GUI-Browsing, which targets GUI-based information-seeking tasks, (2) GUI-General, which covers broader web manipulation tasks, and (3) gameplay, which focuses on lightweight web-based mini-games executed in a browser sandbox.

2.5.1 Task Design

High-quality, sufficiently challenging, and verifiable task data for end-to-end RL remains extremely scarce. In the following, we introduce how to design training tasks that are both diverse in the form and equipped with reliable verification signals.

GUI-Browsing. To enable autonomous exploration in complex reasoning scenarios, we design an automated pipeline for synthesizing large-scale, verifiable GUI-browsing tasks. These tasks are conceptually similar to deep research tasks [43], except that agents must satisfy the information-seeking requirements solely through analyzing screenshots, without access to search APIs. Our synthesis framework includes two main approaches:

(1) Multi-Condition Obfuscation: We begin by extracting core entities and their attribute features from authoritative knowledge sources (e.g., Wikipedia). Each feature is scored for distinctiveness using an LLM. Highly revealing attributes are removed, while the remaining ones are rewritten by the LLM to increase abstraction and reduce specificity. This process produces complex questions defined by multiple indirect constraints, requiring the model to combine and reason over blurred signals in order to identify the correct answer. For example, from a Wikipedia page we generate the following obfuscated question: "Discovered by a representative from the Music And Cabaret talent agency, this group had a founding lineup—initially under another name—that included members from Dreghorn and Irvine, plus a lead guitarist and drummer. The lead vocalist joined after being recommended by a founding member who saw them perform with a

Kilmaurs-based band, and their lead guitarist left to form another ensemble before late 1975. Which record label did this group sign with?"

(2) Multi-Hop Chain-Like Conditions: We begin from an entity's webpage and follow its hyperlinks to identify structurally related entities. For each linked entity, we extract and obfuscate descriptive features, creating tasks where the linked entity becomes the answer. We then treat the linked entity's page as the new starting point and repeat this process recursively, generating tasks for progressively deeper levels. At each step, the answer from the previous hop is embedded within the new question, forming a coherent reasoning chain. Finally, the atomic steps are semantically integrated into a single multi-hop question that requires the model to synthesize intermediate answers, mirroring the layered nature of knowledge propagation online and substantially increasing the demand for deep, sequential reasoning. To ensure difficulty, we filter the synthesized data by discarding instances that can be trivially solved using prior knowledge or a single-turn search, keeping only truly challenging and verifiable tasks for training.

GUI-General. To evaluate general-purpose interaction capabilities, we construct a dataset of GUI-General tasks using an offline synthesis pipeline centered on general websites. We begin by curating candidate websites from public collections, filtering out inaccessible pages, login-gated services, and trivial categories such as static information pages or casual games. For each selected website, VLMs are employed to identify and extract its core functionalities. Based on these, we synthesize tasks at the single-page level through a structured process: removing overly simple functions, composing executable instructions, merging prerequisite sub-tasks, and refining task descriptions for clarity, objectivity, and verifiability. The resulting dataset provides a diverse pool of executable, GUI-interaction-focused tasks that serve as queries for RL training, covering 690 websites across a wide range of domains.

Gameplay. For the game domain, we construct the RL dataset through two complementary sources. First, we collect publicly available HTML5/WebGL mini-games that can run directly in the browser sandbox. Second, to further expand coverage, we synthesize new games using LLMs, which generate lightweight code implementations that preserve core gameplay mechanics while exposing explicit state interfaces. For both real and synthesized titles, we create concise JavaScript verification scripts that query runtime variables (e.g., score, level index, remaining lives) and provide time-aligned state attributes. These observations establish a reliable mapping from agent actions to environment transitions and reward signals. Finally, all interaction records are consolidated into a unified JSON schema containing scalar rewards, termination flags, and metadata (e.g., game version and verification checksums).

2.5.2 Reward Design

A reliable reward system is essential for stable policy optimization, requiring feedback signals that are both consistent and trustworthy across heterogeneous environments. We categorize our reward design based on whether the correctness of an agent's output can be deterministically verified:

Deterministically verifiable tasks. In domains where automatic function-based verifiers are available (e.g., games), we directly compute binary correctness signals as rewards. For GUI-Browsing tasks, where answers can be matched against reference ground truth, we instead employ LLM-as-Judge [20] to evaluate the agent's prediction against the target answer.

Non-verifiable tasks. In more open-ended settings, such as GUI-General tasks, neither formal verifiers nor reference answers exist. To address this, we employ UI-TARS-2 as a generative outcome reward model (ORM) that produces scalar rewards conditioned on the agent's trajectory. The ORM takes as input the full text history together with the last five screenshots (to fit within the context window) and outputs a score indicating task success. To achieve this, we specifically enhance UI-TARS-2's capability of ORM through targeted data annotation and single-turn RL, ensuring that its reward predictions are accurate, consistent, and robust for downstream multi-turn RL.

2.5.3 Asynchronous Agent Rollout via Stateful Environment

Traditional batch-based rollout approaches often become bottlenecked by complex long-tail problems, reducing training efficiency and creating off-policy distribution drift. Our multi-turn RL training infrastructure

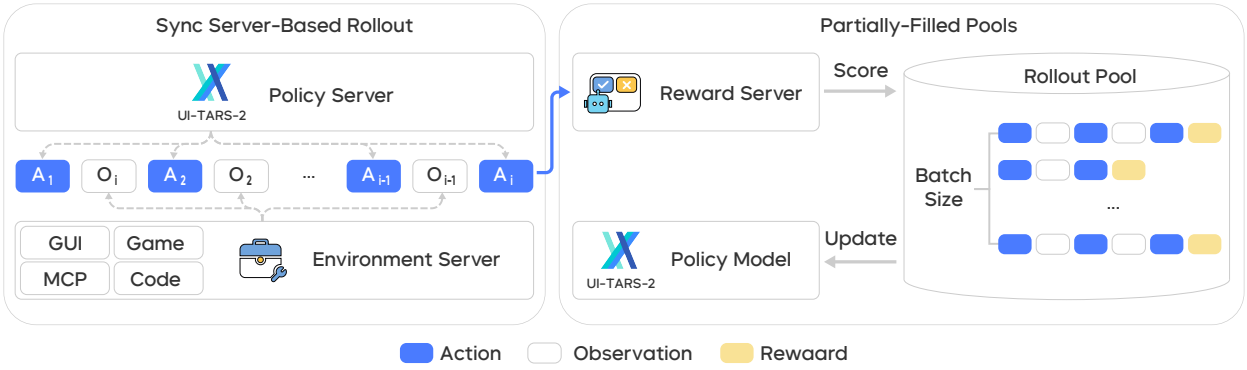


Figure 6 The multi-turn RL training infrastructure of UI-TARS-2.

(Figure 6) is developed for two core objectives: (1) enhancing training stability and (2) optimizing efficiency in multi-turn rollout interactions and training sample organization. UI-TARS-2 implements several key features:

Asynchronous Inference with Server-Based Rollout. We adopt a fully asynchronous inference system utilizing online server-mode processing. By encapsulating policy inference within asynchronous server architecture, we decouple agent reasoning framework implementation from policy inference execution. This design significantly enhances framework usability, which supports easily-developed new agent interaction handlers, while improving model inference efficiency through asynchronous inference.

Streaming Training with Partially-Filled Rollout Pools. Traditional batch-mode rollout requires complete batch inference before training initiation, potentially creating bottlenecks with long-tail cases that delay subsequent training cycles. Our system maintains a dynamic rollout pool where training updates commence once completed traces reach the minimum batch size threshold. Incomplete rollout traces remain in the pool for subsequent training iterations, ensuring continuous learning progress. This feature is conceptually similar to Kimi-Researcher [39].

Stateful Agent Environment Integration. We implement stateful agent environments that preserve execution states across multiple tool invocations, enabling continuous state transitions and maintaining context throughout extended problem-solving sessions. This approach supports complex, multi-step reasoning processes that require persistent environmental memory.

2.5.4 RL Training Algorithm

UI-TARS-2 is trained using Proximal Policy Optimization (PPO), where the policy is updated according to the following objective function:

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D},o_{\leq t}\sim\pi_{\theta_{\text{old}}}(\cdot|q)}\left[\min\left(\frac{\pi_\theta(o_t \mid q, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t \mid q, o_{<t})}\hat{A}_t,\right.\right.$$
$$\left.\left.\text{clip}\left(\frac{\pi_\theta(o_t \mid q, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t \mid q, o_{<t})}, 1-\varepsilon_{low}, 1+\varepsilon_{high}\right)\hat{A}_t\right)\right], \tag{4}$$

where π_θ is policy model, $\pi_{\theta_{\text{old}}}$ is previous policy model.

Following VAPO [83] and VC-PPO [82], UI-TARS-2 integrates several critical enhancements to broaden the exploration space and improve stability, especially in long-horizon settings:

Reward Shaping. To promote more strategic agent behaviors, the reward signal is mainly determined based on the correctness of the final outcome. In certain scenarios, we employ format rewards and length penalties to discourage premature termination or infinite continuation.

Decoupled GAE. To address the challenge of value estimation bias over long sequences, we employ the Decoupled Generalized Advantage Estimation (Decoupled-GAE) [82], allowing the computation of advantage for the policy and value function to use different coefficients. Specifically, we set λ_{policy} and λ_{critic} to be different. This approach prevents decay in the critic's value estimates when dealing with lengthy token sequences, thereby promoting stability during long-horizon training.

Length-Adaptive GAE. To mitigate the issue of inconsistent advantage estimation for sequences of varying lengths, we employ Length-Adaptive Generalized Advantage Estimation (Length-Adaptive GAE) [83] technique, adjusting the GAE parameter λ_{policy} based on the sequence length. Specifically, we set $\alpha = 0.05$ in length-adaptive formula $\lambda_{policy} = 1 - \frac{1}{\alpha l}$ to control the overall bias-variance trade-off.

Value Pretraining. To mitigate the value initialization bias, we adopt the Value-Pretraining [83], which involves offline training of the value model to convergence under a fixed policy. Specifically, responses are sampled continuously from a fixed policy (e.g., π_{sft}), and the value model is updated using GAE with $\lambda = 1.0$ (equivalent to Monte Carlo return), providing stable and reliable optimization. Training continues until crucial metrics such as value loss and explained variance reach sufficiently low levels, indicating effective convergence. The resulting value model checkpoint is then used as the initialization for subsequent experiments, ensuring more accurate and calibrated value estimation from the outset.

Clip Higher. To further promote exploration, we decouple the PPO clipping parameters as recommended by DAPO [80], introducing distinct lower (ε_{low}) and upper (ε_{high}) clipping bounds. Increasing ε_{high} affords greater flexibility for raising the likelihood of low-probability actions, thus enlarging the exploration space. Conversely, ε_{low} is maintained at a low value to avoid prematurely eliminating tokens, which would risk collapsing the diversity of potential outputs.

2.6 Merging Vertical Agents via Parameter Interpolation

A central goal of UI-TARS-2 is to develop a unified digital agent that not only handles structured desktop and web interfaces but also extends to dynamic environments. A natural approach would be to conduct joint RL across all environments and tasks. However, this is challenging in practice: domains differ substantially in action/state spaces, task horizons, and rollout complexity, making large-scale joint optimization unstable and computationally prohibitive. Instead, we adopt a simpler but effective strategy that leverages the observation that models fine-tuned from the same pre-trained checkpoint remain approximately linearly mode-connected in parameter space [48]. This property enables us to train specialized agents independently for different domains and then merge them through parameter interpolation, thereby consolidating their strengths without the cost of multi-domain joint training.

Concretely, starting from a shared SFT initialization, we conduct multiple RL runs tailored to different environments—for example, **GUI-Browsing** tasks focused on information seeking, **GUI-General** tasks covering broader web manipulation, and **Game** environments based on interactive mini-games—alongside additional variants trained on other domains and corresponding tools (e.g., GUI-SDK). We then merge these trained models by interpolating their parameters:

$$\theta^{(\text{merge})} = \sum_{k \in \{\text{GUI-Browsing}, \text{GUI-General}, \text{Game}, \text{GUI-SDK}, \ldots\}} \alpha_k \cdot \theta^{(k)}, \quad \text{s.t.} \quad \sum_k \alpha_k = 1,\ \alpha_k \geq 0, \tag{5}$$

where $\theta^{(k)}$ denotes the parameters of each domain-specialized model. Empirically, this interpolation strategy preserves the performance of each specialized vertical while enabling strong cross-domain generalization. On composite tasks requiring skills from multiple domains, the merged model performs almost comparably to the best specialized model in each relevant domain, without additional optimization cost.

3 Experiments

This chapter presents a comprehensive experimental analysis of UI-TARS-2. Although the training spans multiple domains and tool integrations, we focus our discussion on two representative settings: GUI-based interaction and game environments. These two cases highlight complementary challenges: structured interface operation on the one hand, and dynamic long-horizon control on the other.

3.1 Experimental Setup

UI-TARS-2 is initialized from the pre-trained checkpoint of Seed-thinking-1.6 [11], and leverages all of its post-training data. The architecture includes a 532M-parameter vision encoder and a Mixture-of-Experts (MoE) LLM with 23B active parameters (230B total). Building on this base, we conduct multiple iterative training cycles consisting of SFT, RL, and RFT, progressively refining the model's capabilities.

We conduct evaluations across a diverse set of benchmarks that comprehensively assess agent capabilities:

GUI Benchmarks. We evaluate our model across a diverse suite of benchmarks spanning three categories: computer use, mobile use, and browser use. For **computer use**, OSWorld [75] provides 369 tasks across Ubuntu, Windows, and macOS with detailed configurations and evaluation scripts, while WindowsAgentArena [10] adapts this framework to over 150 Windows-specific tasks. To assess deeper system-level capabilities, we also include TerminalBench [66], which measures proficiency in command-line environments, and SWE-Bench [28], which evaluates repository-level software engineering tasks. For **mobile use**, AndroidWorld [52] offers 116 tasks across 20 mobile applications within a live Android emulator, with dynamic task variations generated via randomized parameters. For **browser use**, Online-Mind2Web [77] contains 300 realistic tasks across 136 websites, while BrowseComp-en [73] and BrowseComp-zh [88] provide high-difficulty multi-hop questions. For the above benchmarks, UI-TARS-2 is allowed to use either GUI operations or GUI SDK.

Game Benchmarks. We develop a **15 Games Collection** from our game pool, which is used to measure in-domain performance. We also leverage an OOD benchmark: **LMGame-Bench** [24], which evaluates LLM agents' game-playing abilities across six classic titles through a unified Gym-style interface, with optional perception and memory scaffolds designed to stabilize vision and long-horizon control. It reports performance under both harnessed and unharnessed settings. For all these collections, evaluations are conducted within a browser-sandboxed, screenshot-only setting. UI-TARS-2 interacts with games through a human-like action space (mouse clicks, key presses, and scrolling), mirroring how players operate in real environments. Results are reported as raw per-game scores as well as the mean normalized score across titles.

Compared Baselines. For GUI benchmarks, we compare UI-TARS-2 against state-of-the-art proprietary models, including Claude 4 [3], OpenAI-o3 [44], and OpenAI CUA-o3 [45], as well as previous UI-TARS variants. For game benchmarks, we evaluate Claude (Computer Use) [2], OpenAI CUA-o3, OpenAI-o3, Gemini-2.5 Pro [14], and Claude 3.7/4 [3].

3.2 Main Results

GUI Main Results. As shown in Table 1, UI-TARS-2 establishes superior performance across a wide range of GUI-agent benchmarks. Compared to previous versions of UI-TARS and other strong baselines such as OpenAI CUA-o3 and Claude 4, our model demonstrates consistent improvements across computer use, mobile use, and browser use settings. In particular, UI-TARS-2 surpasses UI-TARS-1.5 on all reported benchmarks, achieving 47.5% on OSWorld, 50.6% on WindowsAgentArena, 73.3% on AndroidWorld, and 88.2% on Online-Mind2Web, highlighting the benefits of iterative training and reinforcement learning. **Benefits from GUI-SDK**: With the integration of extended SDK functions, UI-TARS-2 is further equipped to handle system-level tasks beyond surface-level GUI interaction. In this setting, the model achieves 45.3% accuracy on Terminal Bench, 68.7% on SWE-Bench, 50.5% on BrowseComp-zh, and 29.6% on BrowseComp-en. For comparison, when restricted to GUI-only operation, the scores on BrowseComp-zh and BrowseComp-en are 32.1% and 7.0%, respectively. This clear performance gap demonstrates that GUI-SDK augmentation enables the model to perform more complex reasoning and tool-use behaviors, equipping UI-TARS-2 with the broader skills expected of general computer-use agents. **OOD Generalization**: Most of the tasks of GUI-Browsing and GUI-General are browser-focused tasks, after RL training, the resulting model exhibits strong OOD generalization. On Online-Mind2Web, RL improves accuracy from 83.7% (the SFT baseline in the final iteration) to 88.2%. More strikingly, the RL-trained model transfers effectively to domains that were not the primary focus of training: for example, OSWorld improves by nearly 10.5% (from 43.0% to 47.5%) and AndroidWorld by over 8.7% (from 64.6% to 73.3%). These results highlight the ability of task-specific RL to induce broadly transferable skills, enabling GUI agents to perform reliably in previously unseen environments.

Game Main Results. As shown in Table 2 and Table 3, UI-TARS-2 demonstrates strong performance across

Table 1 Performance on computer use, mobile use, and browser use benchmarks. "-" indicates unavailable; ✗ denotes lack of ability; and † indicates results obtained with an extended action space that includes GUI-SDK. Terminal Bench results are reported on 75 out of 80 tasks due to compatibility issues with our internal environment. Abbreviations: WAA (WindowsAgentArena), BC-en (BrowseComp-en), BC-zh (BrowseComp-zh), TB (Terminal Bench), SB (SWE-Bench).

	Computer Use				Mobile Use	Browser Use		
Model	**OSWorld**	**WAA**	**TB**	**SB**	**AndroidWorld**	**Online-Mind2web**	**BC-zh**	**BC-en**
Claude-4-Sonnet	43.9	-	39.2	72.7	-	-	22.5	14.7
Claude-4-Opus	-	-	43.2	72.5	-	-	37.4	18.8
OpenAI o3	✗	✗	30.2	69.1	✗	✗	-	49.7
OpenAI CUA-o3	42.9	-	✗	✗	52.5	71.0	-	-
UI-TARS	24.6	-	✗	✗	44.6	-	✗	✗
UI-TARS-1.5	42.5	42.1	✗	✗	64.2	75.8	✗	✗
UI-TARS-2	47.5	50.6	45.3†	68.7†	73.3	88.2	32.1 (50.5†)	7.0 (29.6†)

Table 2 15 Games Collection results. The last row reports the mean normalized score, computed by dividing each game score by the human score and averaging across games.

Game	**Human**	**UI-TARS-2-SFT**	**UI-TARS-2-RL**	**OpenAI CUA**	**Claude Computer Use**
2048	1024.31	968.00	932.40	911.21	800.00
Emoji-sort-master	5.15	2.90	4.50	1.80	1.40
Energy	10.08	2.40	3.30	0.82	1.04
Free-the-key	5.54	0.00	0.70	0.00	0.00
Gem-11	186.85	84.90	63.90	47.00	55.00
Hex-frvr	5276.85	1952.00	2389.00	615.59	523.07
Infinity-Loop	6.58	1.60	6.10	3.30	1.90
Laser-maze-puzzle	17.83	2.70	5.60	1.40	1.40
Maze:Path-of-Light	7.17	1.10	2.00	0.35	0.82
Merge-and-double	790.31	519.00	594.40	102.33	212.70
Shapes	5.42	4.60	5.90	0.90	0.24
Snake-solver	3.92	2.10	3.00	0.23	0.20
Tiles-master	3.75	3.20	3.10	1.47	1.56
Wood-blocks-3d	4646.00	1900.00	2908.00	1814.00	1632.00
Yarn-untangle	19.75	4.30	7.00	5.05	1.56
Mean Normalized Score	100.00	44.27	59.77	24.73	21.61

both in-domain and out-of-domain game benchmarks. On the 15-game in-house suite—where scores are normalized to Human (100)—UI-TARS-2 achieves a mean normalized score of 59.8, reaching nearly 60% of human-level performance on average. This represents a substantial margin over production systems such as OpenAI CUA and Claude Computer Use, outperforming them by +35.0 and +38.2 points, respectively. Notably, the model is already close to human level on several titles, including 2048 (91.0), Infinity-Loop (92.7), Tiles-master (82.7), Snake-solver (76.5), and Merge-and-double (75.2), and even surpasses human performance on Shapes (108.9). On the out-of-domain LMGame-Bench, UI-TARS-2 remains competitive with frontier general-purpose models. For example, it achieves 117.1 on 2048 (within 9% of o3 at 128.2), ranks near the top on Candy Crush (163.2, above o3's 106.0 and second only to Gemini-2.5 Pro at 177.3), and performs strongly on Super Mario Bros. (1783.2 vs. 1955.0 for o3). While performance is weaker on Tetris and Sokoban, reflecting challenges in very long-horizon planning, the overall results show that UI-TARS-2 can transfer effectively to unseen game mechanics and environments.

3.3 Detailed Analyses

In the following, we present the detailed analyses of UI-TARS-2's RL training.

Table 3 LMGame benchmark results (mean ± std over runs over three runs).

Model	Sokoban	Super Mario Bros	Tetris	2048	Candy Crush	Ace Attorney
claude-3.5-sonnet-20241022	0.0 ± 0.0	1540.1 ± 21.7	12.3 ± 2.5	57.8 ± 16.4	17.0 ± 18.1	1.0 ± 0.0
claude-3-7-sonnet-20250219 (thinking)	0.0 ± 0.0	1430.0 ± 162.2	13.0 ± 0.0	114.2 ± 7.2	126.3 ± 69.1	3.0 ± 0.0
gemini-2.5-flash-preview-04-17 (thinking)	0.0 ± 0.0	1540.7 ± 262.4	19.0 ± 4.6	107.4 ± 3.4	97.7 ± 36.1	1.0 ± 0.0
gemini-2.5-pro-preview-05-06 (thinking)	1.0 ± 0.0	1025.3 ± 443.2	12.3 ± 3.1	120.5 ± 3.9	177.3 ± 64.9	8.0 ± 0.0
llama-4-maverick-17b-128e-instruct-fp8	0.0 ± 0.0	786.0 ± 462.6	11.7 ± 1.2	44.6 ± 11.8	32.3 ± 41.4	0.0 ± 0.0
gpt-4.1-2025-04-14	0.0 ± 0.0	1991.3 ± 1018.5	13.0 ± 1.7	94.5 ± 17.0	101.0 ± 120.2	0.0 ± 0.0
gpt-4o-2024-11-20	0.0 ± 0.0	1028.3 ± 656.0	14.7 ± 2.1	70.4 ± 12.5	59.0 ± 54.6	0.0 ± 0.0
o1-2024-12-17	0.0 ± 0.0	1434.0 ± 0.0	13.0 ± 0.0	128.1 ± 20.8	90.0 ± 0.0	3.0 ± 0.0
o3-2025-04-16	2.0 ± 0.0	1955.0 ± 0.0	31.0 ± 0.0	128.2 ± 0.0	106.0 ± 0.0	8.0 ± 0.0
o4-mini-2025-04-16	1.3 ± 0.6	1348.3 ± 178.1	15.0 ± 3.6	97.6 ± 29.2	110.7 ± 49.7	2.0 ± 0.0
UI-TARS-2	0.3±0.0	1783.2±63.7	16.0±1.3	117.1±1.9	163.2±31.3	7.0±0.0

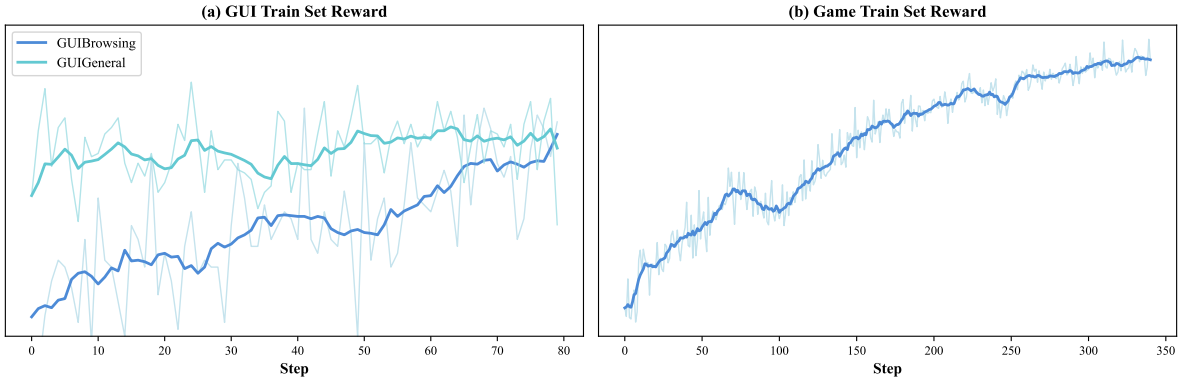


Figure 7 Training reward dynamics for GUI-Browsing, GUI-General, and game scenarios in UI-TARS-2.

Training Rewards and Entropy. As shown in Figure 7, RL on GUI-Browsing, GUI-General, and Game tasks exhibits a clear upward trend in training rewards. For the game domain, we report the average reward over the 15-game suite, with per-game reward curves provided in Figure 13. These results indicate that the policy steadily improves under RL supervision, and the consistency of this trend across both structured GUI tasks and dynamic game environments highlights the general effectiveness of multi-turn RL in diverse settings.

Interestingly, we observe distinct entropy dynamics compared to recent reasoning-focused RL work. Whereas reasoning RL often shows monotonic entropy reduction (reflecting a shift toward deterministic exploitation), our GUI and web-game experiments frequently exhibit rising entropy during training (Figure 8). This suggests that the model maintains or even expands its exploration space as training progresses, enabling it to acquire new interaction patterns rather than collapsing prematurely into narrow exploitation. Such behavior reflects the need for broader exploration in visually rich and highly interactive environments, where diverse strategies are often required for effective learning.

Viability of VLM-as-Verifier. As shown in Figure 7, UI-TARS achieves a steady increase in rewards on both training tasks, with the improvements correlating positively with performance gains on downstream benchmarks (Table 1). Notably, although training was conducted with a generative reward model (for the web operation task) or GPT-4o-as-judge (for the GUI browsing task), manual inspection of rewards did not reveal any substantial signs of reward hacking. This suggests that, unlike in general text generation tasks, employing a VLM-as-verifier for agent RL is feasible—possibly because task completion in agent settings can be defined more concretely and evaluated more objectively.

To more quantitatively examine the potential impact of reward hacking, we constructed an in-house ORM evaluation set containing 300 human-annotated GUI agent traces. On this benchmark, UI-TARS-2 achieved an F1 score of 83.8 as the generative ORM in the binary classification setting, indicating reasonably strong robustness. A closer analysis of misclassified cases reveals that the current ORM still exhibits a relatively high

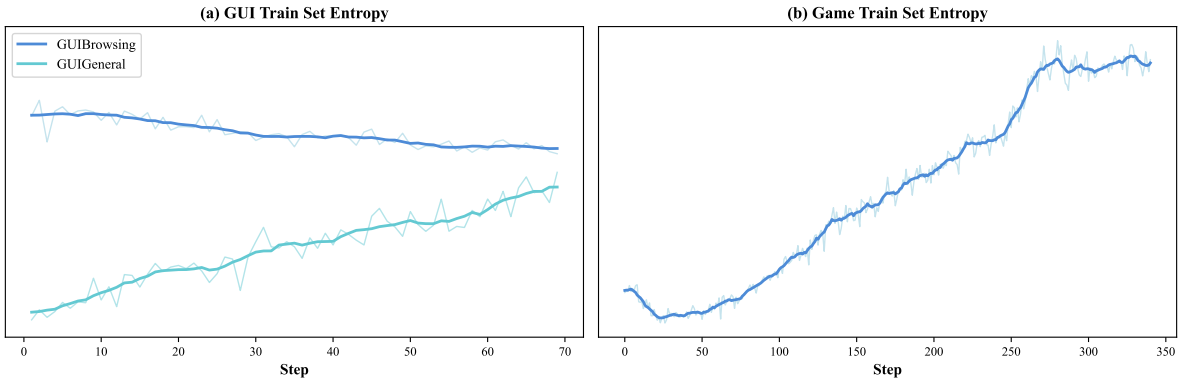


Figure 8 Training entropy dynamics for GUI-Browsing, GUI-General, and game scenarios in UI-TARS-2.

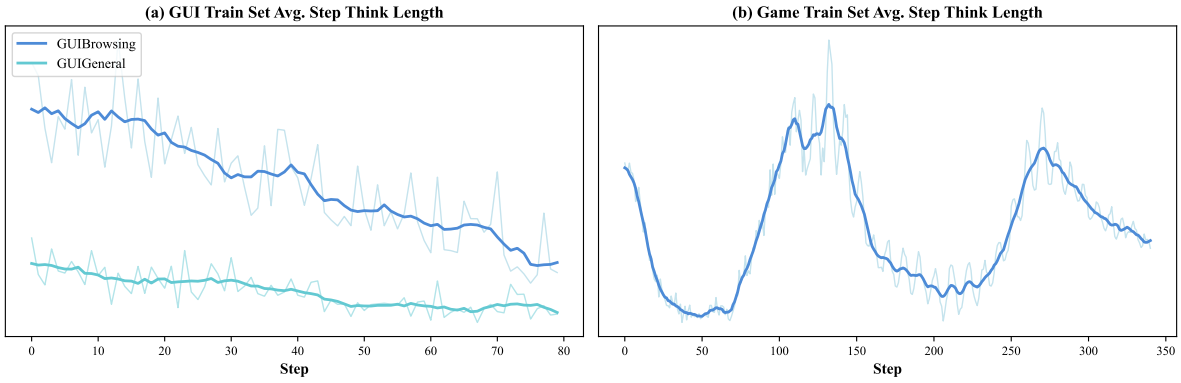


Figure 9 Training dynamics of average step think length for the GUI-Browsing, GUI-General, and game scenarios in UI-TARS-2 RL training.

false positive rate. Nevertheless, even such an "imperfect" ORM proved effective in RL training. We attribute this to the fact that, even in a case where the final task outcome is incorrect, the agent might also execute many correct intermediate steps. In false positive cases, the model still receives appropriate rewards for these correct steps, and these positive contributions outweigh the erroneous rewards given to incorrect actions.

Average Think Length. In our GUI experiments, we observe a consistent decline in the model's average step-level think length as RL training progresses (Figure 9). This trend stands in contrast to common expectations that more complex reasoning processes emerge over time. One possible explanation is that, in GUI tasks, agents primarily make progress through interaction with the environment rather than extended internal reasoning alone. Consequently, once the correct GUI action can be predicted, the agent can obtain rewards directly, reducing the need for longer deliberation.

In the game domain, we observe a periodic pattern in the think length: it tends to increase for a period of time and then gradually decrease. Our analysis suggests that this pattern is tied to the progressive difficulty of the game (which is our design of the training curriculum, gradually increasing the game difficulty). Whenever the agent enters a new level, the increased difficulty requires more reasoning and decision-making to achieve success, leading to a rise in think length. As the agent becomes familiar with the challenges at a given difficulty level, the think length gradually decreases—similar to the trend observed in other GUI tasks—until the next difficulty escalation, at which point the cycle repeats.

Number of Environment Interaction Rounds. We observe that UI-TARS-2's interaction scaling curve—measured by the number of environment interaction rounds—is not always positively correlated with performance. As shown in Figure 10(a), while rewards steadily increase with training steps, the number of steps required

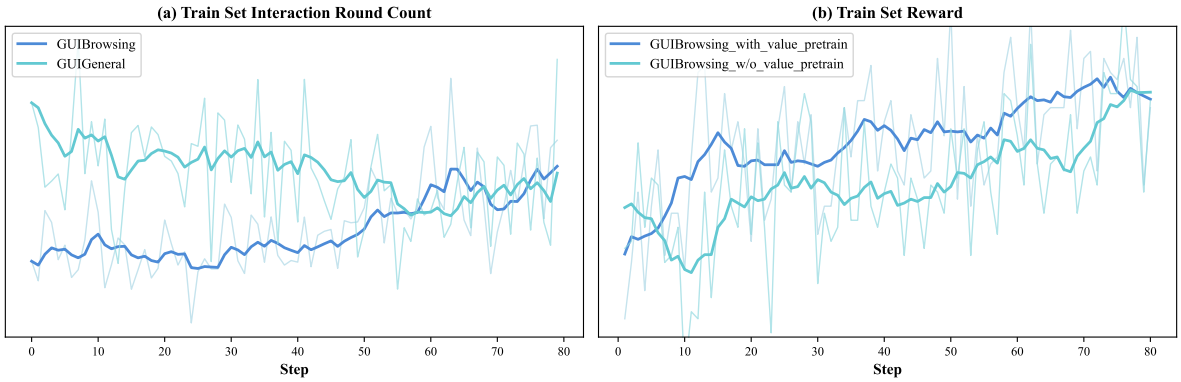


Figure 10 (a) Training dynamics of average interaction round count for the GUI-Browsing and GUI-General scenarios in UI-TARS-2 RL training; (b) Impact of value model pretraining in GUI-Browsing scenarios.

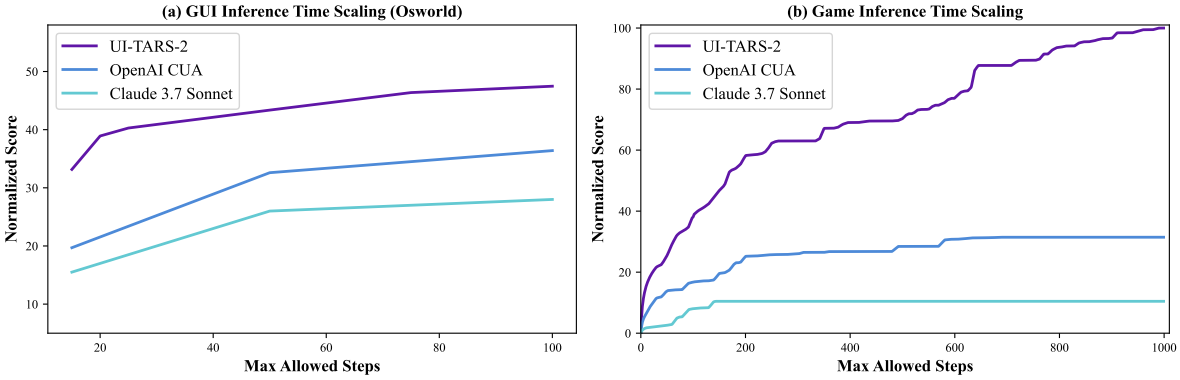


Figure 11 Inference-time scaling evaluation on OSWorld and game benchmarks.

to complete GUI-General tasks gradually decreases. This indicates that, through RL training, the model internalizes task-relevant knowledge and reduces unnecessary exploration, allowing it to solve tasks more efficiently. More broadly, interaction scaling is a common phenomenon in agent RL: models often learn to exploit larger budgets, prolonging trajectories before convergence. Such behavior can be mitigated by explicitly incorporating step budgets into the reward design, encouraging agents to balance efficiency with performance.

Impact of Value Model Pretraining on PPO Training. In our preliminary experimental exploration, we observed that the value estimates of PPO-trained models were often negatively correlated with the obtained rewards. Motivated by this finding, we introduced a value model pretraining stage into the training process. As shown in Figure 10 (b), value pretraining enhances the value model's ability to guide policy learning, leading to consistently higher rewards throughout training.

Inference-time Scaling. The long-horizon nature of gameplay makes it a natural testbed for inference-time scaling. As the allowed step budget increases (Figure 11, our performance curve rises steadily in an almost monotonic, staircase-like pattern, without exhibiting instability spikes. In contrast, baseline curves flatten quickly, indicating a limited ability to translate additional interaction budgets into further gains. The persistent upward trend of our agent, even at step counts orders of magnitude larger than the original budget, suggests that the policy continues to unlock new subgoals as task-specific thresholds are crossed, rather than merely looping or drifting.

We also observe a similarly strong inference-time scaling trend on both OS operation tasks (OSWorld), as shown in Figure 11. The plots clearly indicate that as the maximum allowed inference steps increase, the

Figure 12 Training dynamics training reward of GUI-Browsing and GUI-General during PPO and GRPO training.

model's performance score consistently rises, which serves as a key indicator of its capacity to leverage a larger computational budget for improved outcomes. Interestingly, although RL training has incentivized our agent to complete tasks in fewer steps, the model still exhibits excellent inference-time scaling on the OSWorld evaluation set. This finding highlights that the learned policy retains the flexibility to effectively exploit additional interaction steps at inference time—unlocking further subgoals or exploring alternative solution paths—rather than overfitting to minimal-step strategies during training.

PPO v.s. GRPO. GRPO [57] has been shown to be effective for training across a wide range of reasoning tasks. However, in our preliminary evaluation, we find that PPO consistently outperforms GRPO by a clear margin. As depicted in Figure 12, PPO maintains higher rewards with lower volatility throughout training. To ensure stable learning dynamics and achieve stronger overall performance, we ultimately selected PPO as the optimization algorithm for our main experiments.

Behavior Analyses in Game RL. In the Figure 13, we present the training rewards for every game. Several titles reach or closely approach the human@100-step reference by the end of training (for example, 2048, Infinity-Loop, Emoji-sort-master, Tiles-master, and Shapes, with Shapes surpassing the human reference). A second pattern is from-zero learning: games that the base model could barely play at the start (such as Free-the-key and Yarn-untangle) are trained up to nontrivial scores, indicating a genuine increase in the model's general game-reasoning ability rather than overfitting to a few scripts.

At the same time, a subset of games shows clear plateaus or temporary regressions followed by shallow recovery (for example, Gem-11 and Hex-frvr), suggesting a reasoning ceiling imposed by the starting backbone rather than a lack of optimization steps. The staircase shapes visible on many curves indicate that progress tends to arrive in bursts when task-specific subgoals become reliably attainable; once a subgoal is mastered, learning stabilizes until the next threshold is unlocked.

Overall, the curves imply that additional compute and curriculum can continue to convert training into control gains, but breaking through the remaining walls will likely require stronger long-horizon reasoning and planning capacity (for instance, better credit assignment, curriculum over subgoals, and improved search or memory components), pointing to clear headroom for future scaling.

Analysis of GUI SDK RL. As shown in Figure 14, during GUI-SDK RL, the training score exhibits an overall increasing trend as the training steps progress. This indicates that the model gradually becomes proficient in utilizing external tools to solve the complex problems throughout the training process, seamlessly combining the reasoning and tool call in process. Meanwhile, the training entropy shows a continuous downward trend as training progresses, suggesting that the model's confidence in its predictions steadily improves, leading to enhanced stability and reduced uncertainty in its reasoning path.

Hybrid Agent RL. While parameter interpolation serves as our primary approach for consolidating specialized agents, we also investigated an alternative based on hybrid reinforcement learning. In this setting, we focused on an information-seeking scenario that could be solved through two distinct interfaces: a GUI-only method

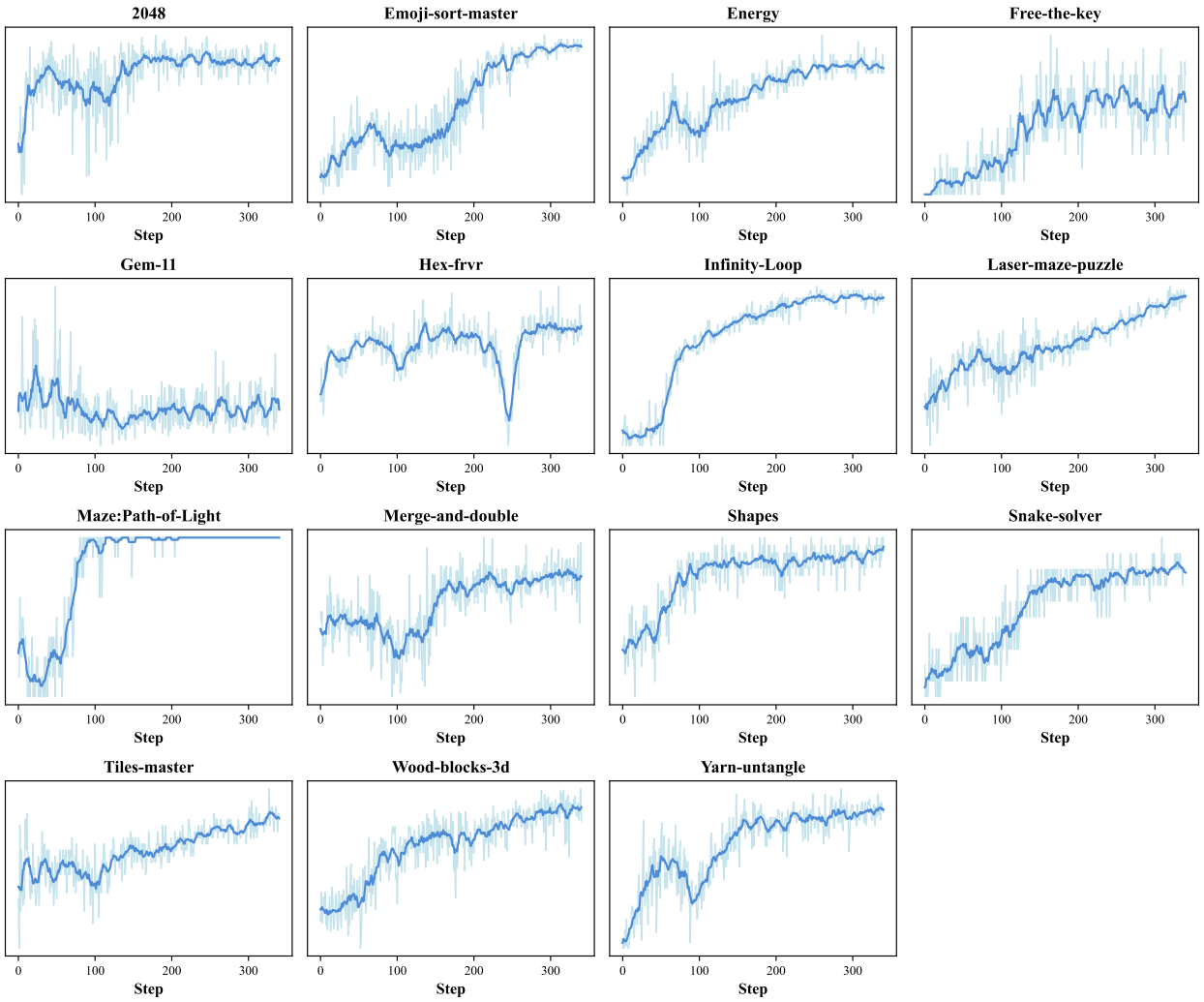


Figure 13 Training dynamics of training rewards for each game in the 15 Games collection.

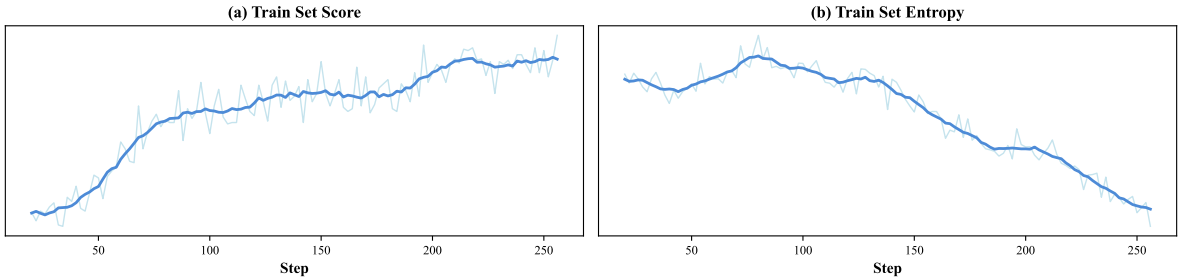
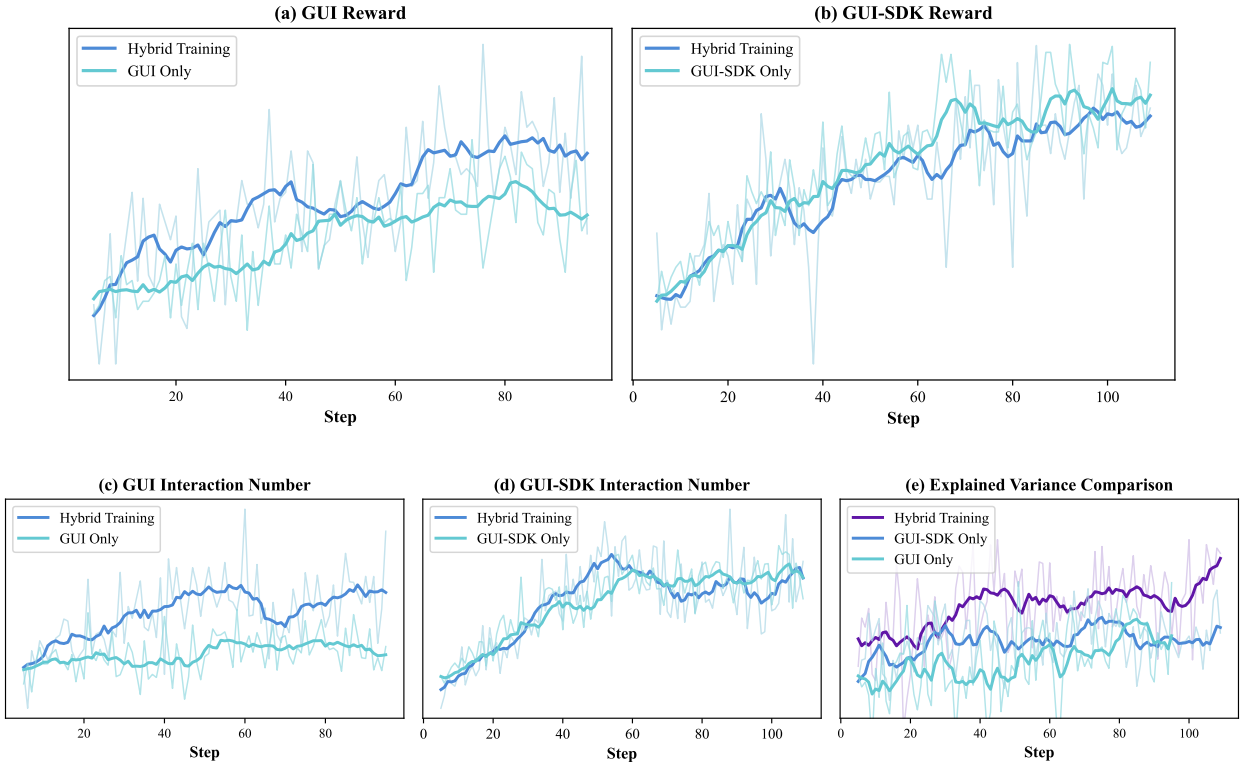


Figure 14 Training dynamics (train set score and entropy) of GUI-SDK RL.

and a GUI-SDK method, where graphical actions are augmented with additional system-level capabilities. The hybrid model was trained to use either interface, whereas baseline models were trained exclusively on a single interface with the same overall batch size.

As shown in Figures 15, hybrid training produced a stronger interaction scaling trend. Even though the training data for each interface was effectively halved compared to the single-interface baselines, the hybrid



(a) Training reward of hybrid training versus training purely on one interface.

Figure 15 (a)(b) Training reward of hybrid training versus training purely on one interface; (c)(d) Iteration length of hybrid training (Orange line) with training purely on one interface (Blue line); (e) Explained Variance of the value model in hybrid training versus training on one modality. Unified tasks enable the value function to perform optimally across all scenarios.

model outperformed the GUI-only baseline when evaluated on pure GUI tasks. This indicates that knowledge acquired through the more capable GUI-SDK interface transfers effectively to GUI-only interaction, boosting competence even in the restricted setting. We further observed that employing a shared value model improved training stability and reward estimation: by learning jointly from trajectories across both interfaces, the value model generalized to a broader range of patterns, yielding higher explained variance than interface-specific baselines (Figure 15(e)).

Compared to parameter interpolation, which merges specialized agents without additional optimization, hybrid training enables more direct cross-interface knowledge transfer but incurs higher training cost. Together, these results highlight two complementary strategies for unifying capabilities across interaction settings: interpolation offers efficiency, while hybrid RL provides stronger transfer.

Quantization for Latency Reduction. We examine the effect of W4A8 quantization on end-to-end efficiency. W4A8 quantization reduces model weights to 4-bit precision and activations to 8-bit precision, enabling faster inference with only a modest loss in accuracy. For UI-TARS-2, quantization increases the token generation rate from 29.6 to 47 tokens/s and reduces the average end-to-end latency per interaction round from 4.0 to 2.5 seconds. On OSWorld, accuracy decreases slightly from 47.5 to 44.4, indicating that the efficiency–performance trade-off remains favorable. These results demonstrate that W4A8 quantization is a practical strategy for deploying GUI agents in latency-sensitive settings while preserving competitive task performance.

4 Related Work

Early LLM-based agents were primarily generic systems driven by prompting recipes that couple reasoning and acting, or by tool-augmented interfaces. Representative examples include ReAct, which interleaves chain-of-thought with environment actions [79], MRKL, a modular neuro-symbolic tool hub [29], and Toolformer, which self-supervises API calls to external tools [54]. In parallel, DeepMind's Gato demonstrated a single multi-task policy acting across diverse embodiments [53]. Building on these ideas, research soon specialized into vertical domains with dedicated benchmarks and interaction environments. For web/GUI interaction, Mind2Web and WebArena provide realistic websites and tasks for web agents [16, 89], while OSWorld enables execution-based evaluation on desktop applications [75]. For software engineering, SWE-bench frames end-to-end repository-level bug fixing and has spawned a series of agentic systems such as SWE-agent with agent–computer interfaces (ACI) [28, 78], and RepoAgent for repository-level documentation and maintenance [36].

GUI Agents. Research on GUI agents has advanced rapidly since the release of early grounding datasets such as ScreenSpot [13]. Within a short time, these datasets reached saturation, prompting a shift of focus from grounding individual elements to developing end-to-end agents capable of performing complete GUI-based tasks. Open-source efforts were the first to drive this transition, with systems such as CogAgent [23], OS-Atlas [74], and Aguvis [76]. This momentum was soon joined by industry initiatives, including OpenAI [46], Anthropic [3], and Bytedance [49], leading to the rapid proliferation of computer-use agents. Early approaches were largely data-driven, relying on diverse human demonstrations for supervised fine-tuning. While these methods enabled the first generation of GUI agents, they suffered from limited generalization and poor robustness in complex environments. More recently, reinforcement learning (RL) has emerged as a promising direction, with systems such as ARPO [35] and Mobile-GUI-R1 [58] applying RL-based training.

Game Agents. Game environments provide a natural testbed for studying interactive decision-making, where long-horizon control and strategic exploration are essential. Digital games have historically been central to AI research due to their complexity, diversity, and controllability. Seminal work ranges from classical board games such as Go [60], to Atari benchmarks [9], to large-scale strategy games like StarCraft II [37], and open-ended environments such as Minecraft [17]. However, a key limitation of these efforts is their specificity: agents were typically optimized for a single game with tailored policies and parameters, hindering generalization across different environments [8, 38, 67].

The emergence of LLMs and VLMs has shifted attention toward more generalist agents [53]. Recent work explores their application to complex game scenarios, such as Pokémon [4, 14]. To cope with the long-horizon and multimodal nature of games, many approaches adopt workflow-style designs, equipping models with explicit modules for memory [68, 72] and planning [59, 81, 86], or fine-tuning VLMs on specific titles for domain specialization [31, 51]. A key distinction lies in the interaction modality. Most existing systems depend on textual observations and predefined semantic actions exposed via game APIs [14, 68]. By contrast, our framework engages with games through the same modality used for GUI tasks—native GUI actions grounded in visual input. This unified interface integrates game environments with general computer-use scenarios, eliminating the need for additional handcrafted modules and highlighting the potential of building agents that generalize across diverse interactive digital settings.

Other Relevant Directions. Beyond GUI and games, two other lines of research provide complementary insights into the design of interactive agents. Protocols such as MCP [1] have introduced standardized mechanisms for flexible tool integration, enabling agents to seamlessly interact with search engines, file parsers, or external APIs. Building on this foundation, recent work has pursued two primary directions: end-to-end reinforcement learning [18, 33, 61, 69], which directly optimizes multi-step reasoning with tool calls, and workflow-based methods [22, 50], which orchestrate tools through scripted procedures but often lack flexibility. Early studies mainly focused on simple tool-enhanced tasks such as HotpotQA or MathQA [19, 32], whereas more recent efforts have introduced harder benchmarks like BrowseComp [73], where information is deliberately obfuscated across websites. These benchmarks reveal the limitations of models trained only on simple data [62] and have motivated research on synthesizing high-difficulty datasets [30] and building multi-agent or planning-based systems [12, 26].

In parallel, LLM-based code agents have reshaped software automation. SWE-bench [28] established a

benchmark for repository-level issue resolution and inspired systems such as SWE-agent [78]. This was followed by richer datasets including SWE-Gym [47], SWE-Bench-Extra [6], SWE-ReBench [7], and Multi-SWE-RL [84], which broaden task coverage and programming language diversity. Frameworks such as OpenHands [71] advanced sandboxed agentic coding with execution feedback, while Terminal Bench [66] emphasized command-line proficiency as a critical skill. Alongside, both proprietary models (Gemini [14], Claude [5], GPT-5 [42]) and open-source efforts (Qwen-3 Coder [65], Kimi-K2 [64], GLM-4.5 [85]) have increasingly prioritized agentic coding capabilities. These trends illustrate how reinforcement learning, interactive feedback, and curated datasets are becoming central for scaling code agents.

5 Conclusion

In this work, we presented UI-TARS-2, a native GUI-centered agent model designed to handle both structured computer-use tasks and dynamic, game-like interactive environments. The model is trained through an iterative pipeline that combines multi-turn reinforcement learning, supervised fine-tuning, rejection sampling, and continual pre-training, enabling continual improvement across heterogeneous domains. Our experiments show that while domain-specialized variants can achieve peak scores on individual benchmarks, UI-TARS-2 attains balanced and competitive performance across GUI, browser, mobile, and game tasks within a single unified system. Beyond benchmark results, our analysis (e.g., training dynamics and interaction scaling) yields practical insights into multi-turn agent RL. We also demonstrate that training on diverse environments promotes parameter sharing and capability transfer, giving rise to hybrid skills that integrate graphical interaction with more complex forms of reasoning and decision-making. Taken together, UI-TARS-2 represents a step toward more capable, reliable, and versatile computer-use agents, offering both empirical evidence and methodological principles to guide future research.

References

[1] Anthropic. Introducing the model context protocol, 2024. URL https://www.anthropic.com/news/model-context-protocol.

[2] Anthropic. Developing a computer use model. https://www.anthropic.com/news/developing-computer-use, 2024. Product announcement.

[3] Anthropic. Claude 3.7 sonnet system card. 2025.

[4] anthropic. Claude's extended thinking, 2025. URL https://www.anthropic.com/news/visible-extended-thinking.

[5] anthropic. Introducing claude 4, 2025. URL https://www.anthropic.com/news/claude-4.

[6] Ibragim Badertdinov, Maria Trofimova, Yury Anapolskiy, Sergey Abramov, Karina Zainullina, Alexander Golubev, Sergey Polezhaev, Daria Litvintseva, Simon Karasik, Filipp Fisin, Sergey Skvortsov, Maxim Nekrashevich, Anton Shevtsov, and Boris Yangel. Scaling data collection for training software engineering agents. Nebius blog, 2024.

[7] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents, 2025. URL https://arxiv.org/abs/2505.20411.

[8] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. Advances in Neural Information Processing Systems, 35:24639–24654, 2022.

[9] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of artificial intelligence research, 47:253–279, 2013.

[10] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale. September 2024.

[11] ByteDance. Seed-thinking-1.6, 2025. URL https://seed.bytedance.com/zh/seed1_6.

[12] Zehui Chen, Kuikun Liu, Qiuchen Wang, Jiangning Liu, Wenwei Zhang, Kai Chen, and Feng Zhao. Mindsearch: Mimicking human minds elicits deep ai searcher, 2024. URL https://arxiv.org/abs/2407.20183.

[13] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. arXiv preprint arXiv:2401.10935, 2024.

[14] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv preprint arXiv:2507.06261, 2025.

[15] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. arXiv e-prints, pages arXiv–2409, 2024.

[16] Xiang Deng, Kelvin Guu, Panupong Pasupat, Afra Akyürek, Sheng Zhuang, Wenlong Chen, Tatsunori Hashimoto, Kelvin Guu, and Percy Liang. Mind2web: Towards a generalist agent for the web. In NeurIPS Datasets and Benchmarks, 2023. URL https://arxiv.org/abs/2306.06070.

[17] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. Advances in Neural Information Processing Systems, 35:18343–18362, 2022.

[18] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025. URL https://arxiv.org/abs/2504.11536.

[19] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving, 2024. URL https://arxiv.org/abs/2309.17452.

[20] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. arXiv preprint arXiv:2411.15594, 2024.

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

[22] Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, Xu Shi, Tieqiao Zheng, Liangfan Zheng, Bo Zhang, Ke Xu, and Zhoujun Li. Owl: A large language model for it operations, 2024. URL https://arxiv.org/abs/2309.09298.

[23] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14281–14290, 2024.

[24] Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P. Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-bench: How good are llms at playing games?, 2025. URL https://arxiv.org/abs/2505.15146.

[25] Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, et al. Os agents: A survey on mllm-based agents for general computing devices use. arXiv preprint arXiv:2508.04482, 2025.

[26] Lisheng Huang, Yichen Liu, Jinhao Jiang, Rongxiang Zhang, Jiahao Yan, Junyi Li, and Wayne Xin Zhao. Manusearch: Democratizing deep search in large language models with a transparent and open multi-agent framework, 2025. URL https://arxiv.org/abs/2505.18105.

[27] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. arXiv preprint arXiv:2412.16720, 2024.

[28] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? arXiv preprint arXiv:2310.06770, 2023.

[29] Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. arXiv preprint arXiv:2205.00445, 2022.

[30] Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. Websailor: Navigating super-human reasoning for web agent, 2025. URL https://arxiv.org/abs/2507.02592.

[31] Muyao Li, Zihao Wang, Kaichen He, Xiaojian Ma, and Yitao Liang. Jarvis-vla: Post-training large-scale vision language models to play visual games with keyboards and mouse. arXiv preprint arXiv:2503.16365, 2025.

[32] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models, 2025. URL https://arxiv.org/abs/2501.05366.

[33] Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl, 2025. URL https://arxiv.org/abs/2503.23383.

[34] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, et al. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. arXiv preprint arXiv:2504.01990, 2025.

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

[36] Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. arXiv preprint arXiv:2402.16667, 2024. URL https://arxiv.org/abs/2402.16667.

[37] Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Runji Lin, Yuqiao Wu, Jun Wang, and Haifeng Zhang. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. Advances in Neural Information Processing Systems, 37:133386–133442, 2024.

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015.

[39] MoonshotAI. Kimi-researcher: End-to-end rl training for emerging agentic capabilities. https://moonshotai.github.io/Kimi-Researcher/, 2025.

[40] Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, et al. Gui agents: A survey. arXiv preprint arXiv:2412.13501, 2024.

[41] OpenAI. OpenAI: Introducing ChatGPT, 2022. URL https://openai.com/blog/chatgpt.

[42] OpenAI. Introducing gpt 5, 2025. URL https://openai.com/index/introducing-gpt-5/.

[43] OpenAI. Introducing deep research - openai. https://openai.com/index/introducing-deep-research/, 2025.

[44] OpenAI. Openai o3 and o4-mini system card. https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf/, 2025.

[45] OpenAI. Computer-using agent (cua). https://openai.com/index/computer-using-agent/, 2025. Research preview / blog.

[46] openai. Operator, 2025. URL https://openai.com/index/introducing-operator/.

[47] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym, 2025. URL https://arxiv.org/abs/2412.21139.

[48] Yujia Qin, Cheng Qian, Jing Yi, Weize Chen, Yankai Lin, Xu Han, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Exploring mode connectivity for pre-trained language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 6726–6746, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.451. URL https://aclanthology.org/2022.emnlp-main.451/.

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

[50] Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, Xing Zhou, Dongrui Liu, Ling Yang, Yue Wu, Kaixuan Huang, Shilong Liu, Hongru Wang, and Mengdi Wang. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025. URL https://arxiv.org/abs/2505.20286.

[51] Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, et al. Scaling instructable agents across many simulated worlds. arXiv preprint arXiv:2404.10179, 2024.

[52] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2024. URL https://arxiv.org/abs/2405.14573.

[53] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. arXiv preprint arXiv:2205.06175, 2022.

[54] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761, 2023.

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

[56] ByteDance Seed. Ui-tars-1.5. https://seed-tars.com/1.5, 2025.

[57] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

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

[59] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems, 36:8634–8652, 2023.

[60] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. nature, 550(7676):354–359, 2017.

[61] Huatong Song, Jinhao Jiang, Wenqing Tian, Zhipeng Chen, Yuhuan Wu, Jiahao Zhao, Yingqian Min, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher++: Incentivizing the dynamic knowledge acquisition of llms via reinforcement learning. arXiv preprint arXiv:2505.17005, 2025.

[62] Shuang Sun, Huatong Song, Yuhao Wang, Ruiyang Ren, Jinhao Jiang, Junjie Zhang, Fei Bai, Jia Deng, Wayne Xin Zhao, Zheng Liu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. Simpledeepsearcher: Deep information seeking via web-powered reasoning trajectory synthesis. CoRR, abs/2505.16834, 2025. doi: 10.48550/ARXIV.2505.16834. URL https://doi.org/10.48550/arXiv.2505.16834.

[63] Fei Tang, Haolei Xu, Hang Zhang, Siqi Chen, Xingyu Wu, Yongliang Shen, Wenqi Zhang, Guiyang Hou, Zeqi Tan, Yuchen Yan, et al. A survey on (m) llm-based gui agents. arXiv preprint arXiv:2504.13865, 2025.

[64] Kimi Team. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

[65] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[66] The Terminal-Bench Team. Terminal-bench: A benchmark for ai agents in terminal environments, Apr 2025. URL https://github.com/laude-institute/terminal-bench.

[67] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. nature, 575(7782):350–354, 2019.

[68] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291, 2023.

[69] Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently, 2025. URL https://arxiv.org/abs/2504.14870.

[70] Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, et al. Gui agents with foundation models: A comprehensive survey. arXiv preprint arXiv:2411.04890, 2024.

[71] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025. URL https://arxiv.org/abs/2407.16741.

[72] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2024.

[73] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. arXiv preprint arXiv:2504.12516, 2025.

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

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

[76] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. arXiv preprint arXiv:2412.04454, 2024.

[77] Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. 2025. URL https://arxiv.org/abs/2504.01382.

[78] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

[79] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR), 2023.

[80] 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, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

[81] Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. arXiv preprint arXiv:2303.16563, 2(5), 2023.

[82] Yufeng Yuan, Yu Yue, Ruofei Zhu, Tiantian Fan, and Lin Yan. What's behind ppo's collapse in long-cot? value optimization holds the secret, 2025. URL https://arxiv.org/abs/2503.01491.

[83] Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Ru Zhang, Xin Liu, Mingxuan Wang, Yonghui Wu, and Lin Yan. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks, 2025. URL https://arxiv.org/abs/2504.05118.

[84] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-swe-bench: A multilingual benchmark for issue resolving, 2025. URL https://arxiv.org/abs/2504.02605.

[85] Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. arXiv preprint arXiv:2508.06471, 2025.

[86] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, et al. Proagent: building proactive cooperative agents with large language models. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 17591–17599, 2024.

[87] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, et al. Large language model-brained gui agents: A survey. arXiv preprint arXiv:2411.18279, 2024.

[88] Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, et al. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. arXiv preprint arXiv:2504.19314, 2025.

[89] Shuyan Zhou, Daniel Wu, Tatsunori Hashimoto, Nathanael Schärli, et al. Webarena: A realistic web environment for building autonomous agents. arXiv preprint arXiv:2307.13854, 2023.

6 Contributions

The authors are listed alphabetically by first name, with some names corresponding to internal aliases used within the company.

Algorithm

Core Contributors

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

Contributors

Bo Li
Chen Dun
Chong Liu
Fuxing Leng
Hanbin Wang
Hao Yu
Haobin Chen
Hongyi Guo
Jing Su
Jingjia Huang
Kai Shen
Kaiyu Shi
Lin Yan
Peiyao Zhao
Pengfei Liu
Qinghao Ye
Renjie Zheng
Wayne Xin Zhao
Wen Heng
Wenhao Huang
Wenqian Wang
Xiaobo Qin
Yi Lin
Youbin Wu
Zehui Chen
Zihao Wang

Infra

Core Contributors

Baoquan Zhong
Xinchun Zhang
Xujing Li
Yuanfan Li
Zhongkai Zhao

Contributors

Chengquan Jiang
Faming Wu
Haotian Zhou
Jinlin Pang
Li Han
Qianli Ma
Siyao Liu
Songhua Cai
Wenqi Fu
Xin Liu
Zhi Zhang

Data

Core Contributors

Bo Zhou
Guoliang Li
Jiajun Shi
Jiale Yang
Jie Tang
Li Li
Taoran Lu
Woyu Lin
Xiaokang Tong
Xinyao Li
Yichi Zhang
Yu Miao
Zhengxuan Jiang
Zili Li
Ziyuan Zhao

Contributors

Chenxin Li
Dehua Ma
Feng Lin
Ge Zhang
Haihua Yang

Hangyu Guo
Hongda Zhu
Jiaheng Liu
Junda Du
Kai Cai
Kuanye Li
Lichen Yuan
Meilan Han
Minchao Wang
Shuyue Guo
Tianhao Cheng
Xiaobo Ma
Xiaojun Xiao
Xiaolong Huang
Xinjie Chen
Yidi Du
Yilin Chen
Yiwen Wang
Zhaojian Li
Zhenzhu Yang
Zhiyuan Zeng

Application

Chaolin Jin
Chen Li
Hao Chen
Haoli Chen
Jian Chen
Qinghao Zhao

Supervisor

Guang Shi